PRACTICAL PROGRAMMING IN BASIC
Study Unit 4
24704
Ed 1

PROGRAM DESIGN - THE GAME PLAN

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 4

PROGRAM DESIGN — THE GAME PLAN

Edition 2

STUDY UNIT 4

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

STUDY UNIT 4
PROGRAM DESIGN—THE GAME PLAN

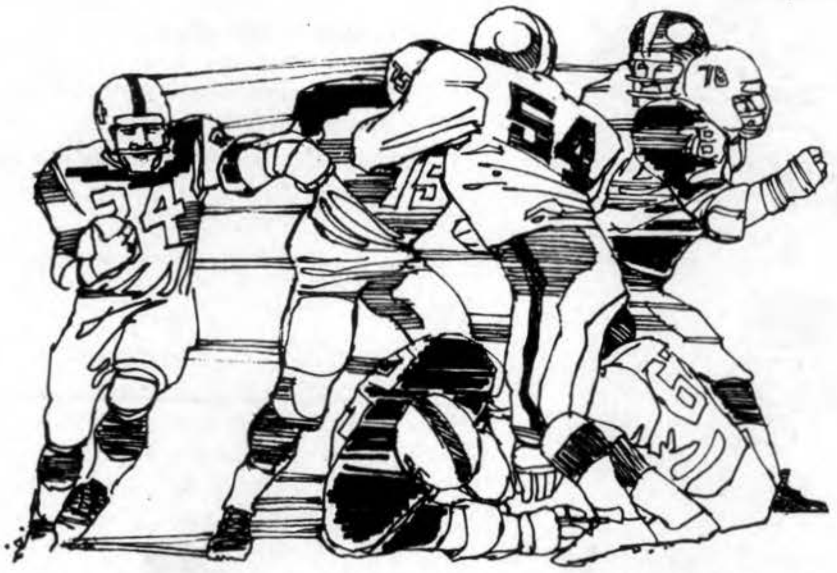


FIGURE 1—The game plan—like a program design—shows each step and sequence of actions to achieve an objective.

DO YOU KNOW?

- **How to read and design a flowchart of a program?**
- **What a decision box is?**
- **What is the initialization routine?**

DESIGNING TO WIN!

Anyone who has ever watched a professional basketball, football or soccer team in action is aware of the fine precision and coordination of every team member. Every person on the team has certain responsibilities and a position to play. If one player does not perform the assigned role, the success of the team is threatened.

FIGURE 2—The result of countless hours of designing and testing is a game plan which will be tested under "real-world" conditions.

Most of the play designs are done in a small office using a paper and pencil or blackboard and chalk. They are sketched and reviewed. They are erased, changed, adjusted, and sketched again.

And although the play designs look good on paper, they must be tried out in practice. The team must run through them and see whether they work or not. Even then, nobody can be sure whether the plays will actually work or not until they are tested under real-world conditions.

To perform as a polished, synchronized unit takes hours and hours of practice. Each player must learn what to do and how to work in harmony with every other member on the team. Equally important, however, are the game plans and special plays used by the team. Without perfectly conceived plans, the very best team in the world cannot perform well or succeed.

Few spectators appreciate the countless hours coaches and scouts spend designing defenses and offenses which are intended to be used in a game lasting no more than one hour. A head coach and assistants may spend several days working out just one or two plays which will be used in a special situation.

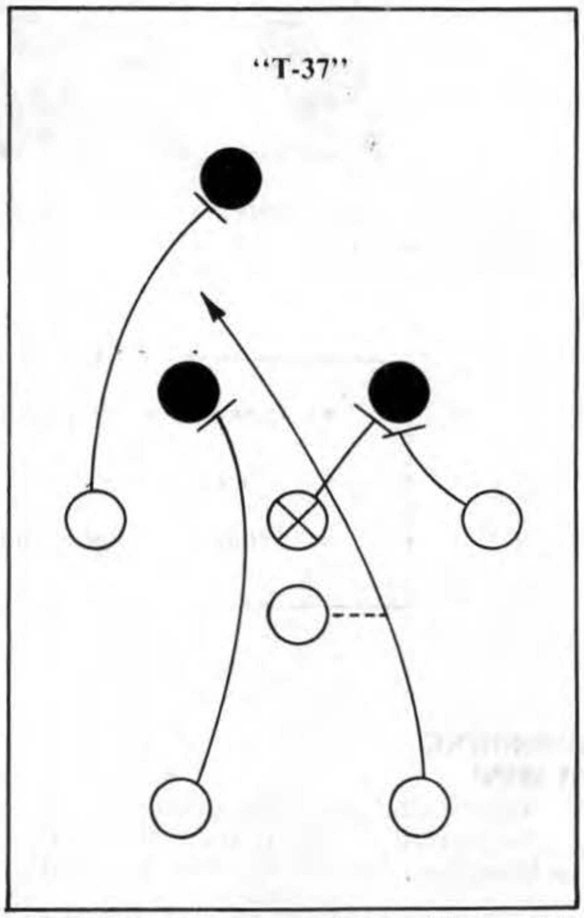


FIGURE 3—"It looks good, but will it work?" Programmers and coaches ask the same question!

The results of actual competition separate successful plays from those which fail. The successful ones are put into the team's "play book" and are used whenever the situation requires them. And certain plays are saved for only very special occasions such as one minute to go and down by two points!

What happens to plays which fail? Are the diagrams thrown away? Most wise coaches maintain a large file of "what doesn't work" so they won't spend valuable time fooling with plans which have already been proven impractical. This sort of documentation is essential. And who knows, maybe some time in the future the coach will realize why the play failed and can make an adjustment so it will succeed.

Being a programmer is very much like being a professional team coach. You are faced with certain very real challenges and it is up to you to find the solutions. Using pencil, template, and worksheet, you design game plans to try out on the computer. Some of these plans will work and some will not. Then, the challenge is to figure out how to make them succeed or file the unsuccessful diagrams for future reference.

Certain programs you will design are intended for use in only one situation, one time. Others are designed for utility purposes and can perform a variety of tasks. But just because it works well in one situation doesn't mean that it will work in all others. Therefore, another challenge of the programmer is to see whether a program can be applied to different circumstances. In other words, parts of the game plan must work no matter what the challenge is. How do we create successful game plans? Let's find out.

PROGRAMS SOLVE PROBLEMS

Now that we are familiar with a computer system, how do we, as programmers, get it to work for us? Computer programs begin as problems, such as:

- How can I keep track of my monthly budget?
- How can my company meet its weekly payroll?
- What will the square footage of a house be if its dimensions are increased by 10% or 15%?

When completed, a computer program can provide solutions to these problems. The key to a successful, accurate solution lies in adherence to the steps of the program development cycle:

1. Analysis
2. Design
3. Coding
4. Testing and debugging
5. Documentation

Let's take a look at each of these steps.

Analysis—"*What's the problem?*"

It is unlikely that a program will work unless careful thought is given to the problem. In the analysis phase of the development cycle, we must determine what needs to be done and what data is required. Questions must be asked. What is the input and how is it to be collected? What is the output, who needs it, and how should it appear?

In the analysis phase, we need no paper, no pen—we must merely use our brains. The failure of many computer programs, costing companies millions of dollars, can often be blamed on faulty or incomplete analysis of the problem.

Certainly, we would never set out on a trip without knowing the destination, the starting point, and the limitations of our automobile. Yet, there are some programmers who, in a hurry to complete their assignment, rush to the solution without thinking of where they are going. That's not to say that the solution will not eventually be achieved. However, research has shown that the solution will be found much more efficiently when the problem is given proper thought at the outset. All aspects of the problem must be clearly identified.

Design—"*How do we proceed?*"

Once we know what the problem is, it is almost mandatory that we put down, on paper, a step-by-step sequence of events that will lead us to a solution. Using the analogy of our car trip, the design phase would include:

- When are we going to leave?
- What roads will we take?
- Where will we make our turns?
- How many stops will we have to make?, etc.

It is in the design phase that we begin to sketch out the plan to be followed. We use worksheets, pencil, flowcharts, and certain procedures which are standardized for all programming.

Coding—"*How to tell the computer?*"

Once we know our design, we must reduce our step-by-step solution to a series of instructions that the computer is capable of doing in a language the computer can understand (such as BASIC).

You will find that coding the program will be relatively quick and accurate if enough time was spent on the analysis and design phases. If not, many of our steps will merely be "guesses."

Testing and Debugging—"*Is the program accurate and reliable?*"

Regardless of how sure we are that our program will work, we must put it to the acid test. Will the computer run the program? If it does work, will it work accurately and in all cases? It is possible, and likely, that some small errors have crept into our program. If so, we must carefully reanalyze, redesign, and retest our program until it works perfectly every time!

Documentation—"*What did we do?*"

This final phase of program development is of equal importance to the other four phases. We must leave a written record so that we, or others, can easily examine or modify the program at a later date. Typically, this is the least appreciated and most abused phase. Programmers tend to get so wrapped up in their work that they are sure they will never forget exactly what they have done. And yet, you will find that a week after you have completed a project you will remember little of your previous work! Some programmers, after a short amount of time, will not even recognize that a program they produced is even theirs!

Maintain a copy of the design and a short narrative in English, of the program. Then, it will take just a matter of minutes to refresh your memory whenever you want to make changes in your program.

THE FLOWCHART

A flowchart is a "road map." It uses symbols to graphically illustrate the sequence that must be followed to produce an accurate program. Flowcharting has its own set of rules just like a program—but unlike a program, it is not done by a computer.

The proper "tools" for neat and accurate flowcharting are a printed worksheet and a template (you can do it "freehand," however). And—use a pencil with eraser (Programmers do *not* use pens).

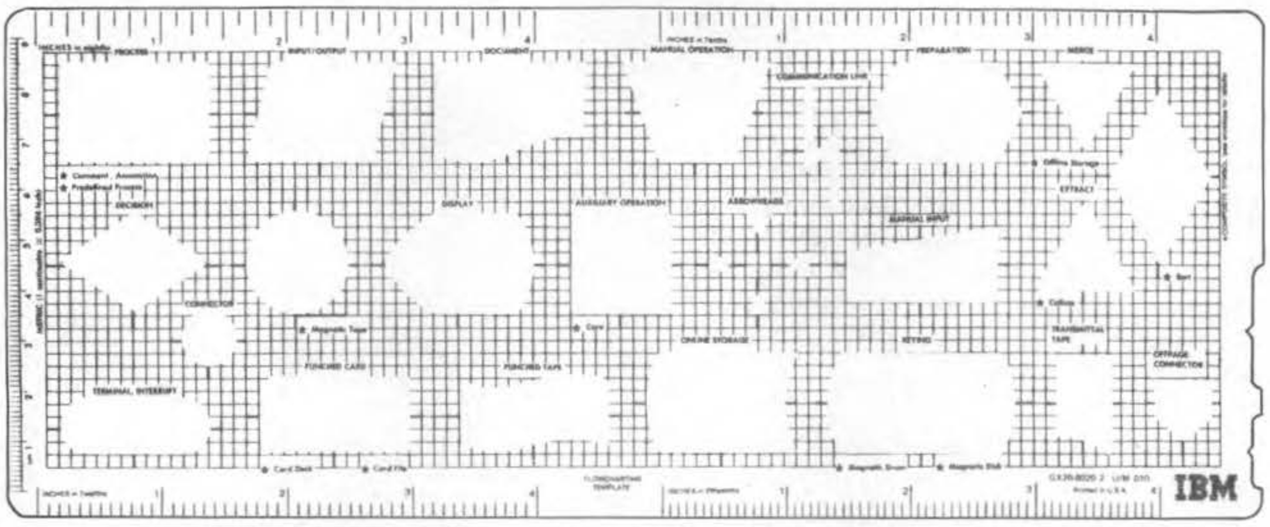
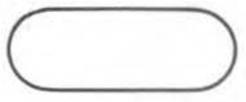


FIGURE 4—This is an illustration of a plastic template used by programmers and task analysts when creating flowcharts. The sequence of operations can be easily depicted by tracing the appropriate symbols onto paper.

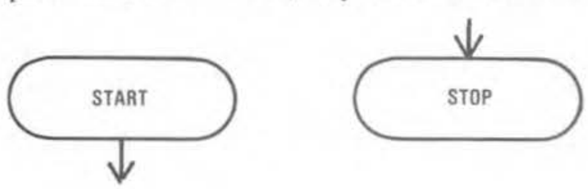
THE TEMPLATE

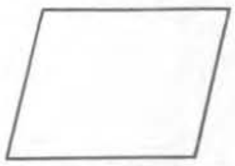
The template is made of hard plastic so that you can neatly draw the various symbols, straight lines and arrows of a flowchart. The symbols are shapes that indicate what type of processing is to be done—the lines and arrowheads show the sequence of events. Inside the symbols, we specify exactly what is to be done in each step.

Let's take a closer look at the symbols we will use. (Not all of the symbols on your template will be discussed at this point—some of them are not used for program design and will be reviewed later.)

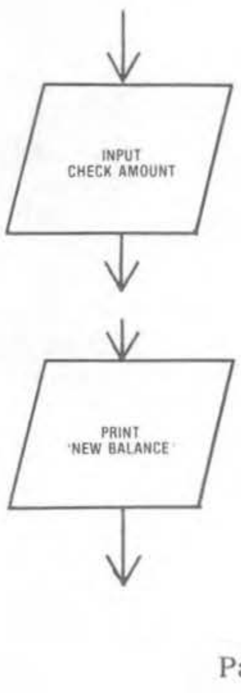


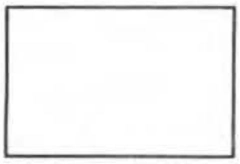
The *terminal* is an oval-shaped symbol used to indicate the beginning and ending point of each program. Every program can only have one starting point and one stopping point. In our flowchart, they will look like this:



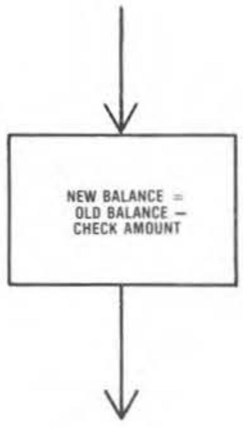


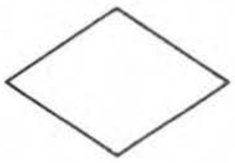
The *input/output* symbol is used whenever data must be read into computer memory from an input device or written from computer memory onto an output device. Examples:





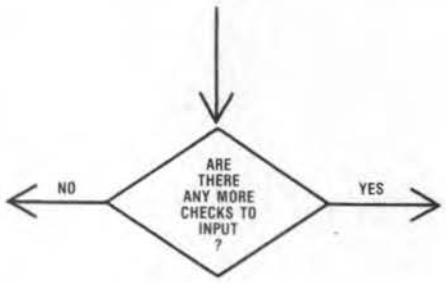
The *processing box* describes the steps that will take place within the CPU, such as arithmetic. It is the box most often drawn on a flowchart. When in doubt, this is the symbol to use more often than not. Example:

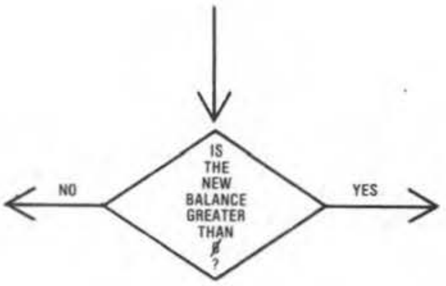


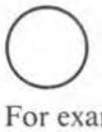


The *diamond-shaped* decision box illustrates the alternative paths that the program allows. Inside this symbol a question is posed. Depending on the answer to this question, only one of the outgoing routes will be taken. (NOTE: These paths must be labeled according to the possible answers to the question.)

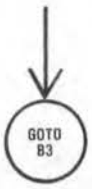
The questions must be phrased in such a way that a computer is capable of answering them. Computer logic (comparisons) is not nearly as complex as human logic. A computer can answer "yes" or "no" when two values are compared—that is, equal to or not equal to. (A computer can also determine which value is greater than—or less than—another.) Examples:







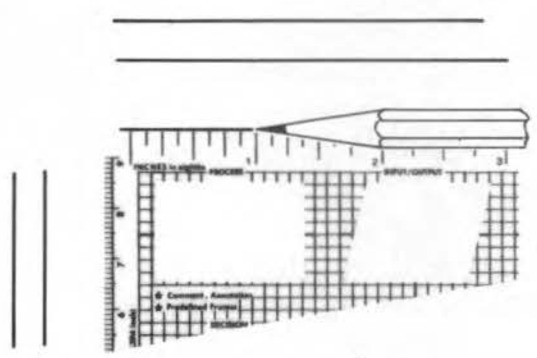
A *circle* is used as a connector symbol. A connector "points" to the instruction to be performed. For example, a connector would be used if we wished to repeat a series of steps (a loop) or to bypass steps under some special circumstances. Inside the connector we write the location, on our worksheet, of the next instruction. Example:

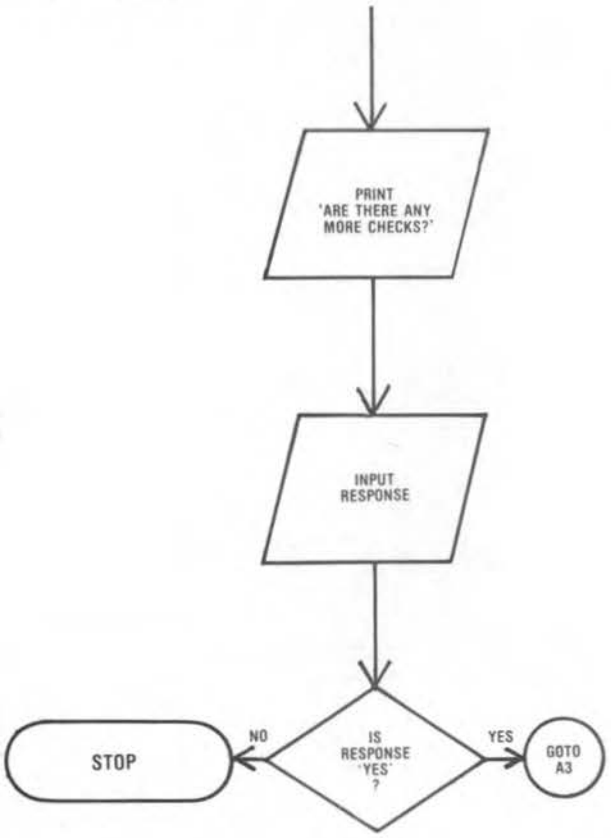


These five symbols are all we need to design the most complex programs!

In order to connect the symbols to each other, flow lines are used. Use the straight edge of your template to produce neat, even lines, such as:

EXAMPLE



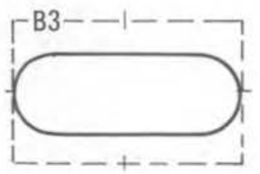


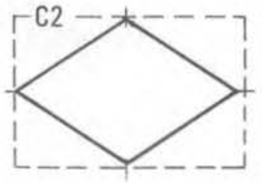
Note the use of arrowheads at the end of each instruction. These arrows ensure that the direction of the sequence is accurately followed. Remember, this is a "flow" chart—just as a river flows, so our design must have a direction. (Your template includes arrowhead symbols, but a good, steady hand can usually draw them clearly without the template.)

THE FLOWCHARTING WORKSHEET

While any clean sheet of paper can be used to draw a flowchart, a worksheet is the easiest to use.

The worksheet has 40 evenly spaced rectangular boxes divided into five columns of 8 rows each. Each box has a unique identifier, placed in its proper column and row (A1 through H5). Your template has a grid work printed on it which makes it simple to center each symbol in its own box. Examples:





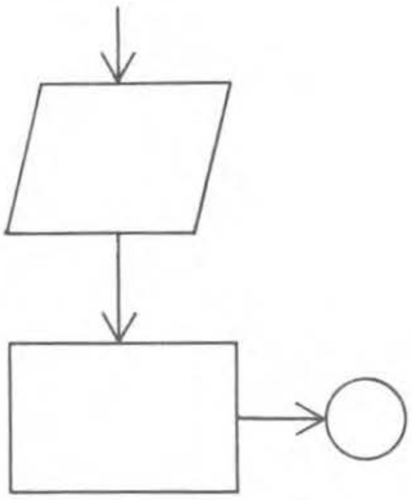
On the top of each worksheet, spaces are provided so that you can write your name, the date, and other information relating the flowchart to the program.

FLOWCHARTING CONVENTIONS

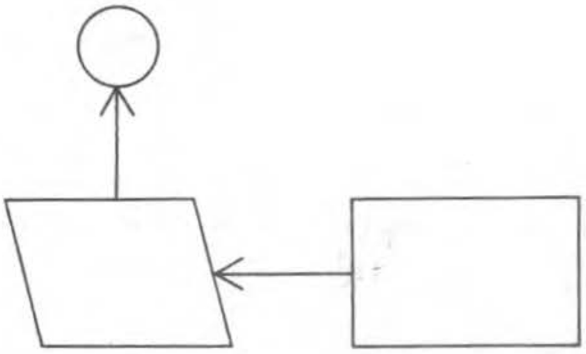
When flowcharting, certain conventions should be followed (we call them "conventions" rather than rules, because the computer will not be able to "read" our flowchart regardless of how we write it).

1. Whenever possible, the flow of our program should be from top to bottom and from left to right.

PROPER



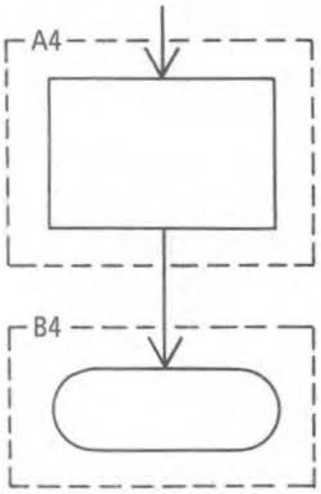
IMPROPER



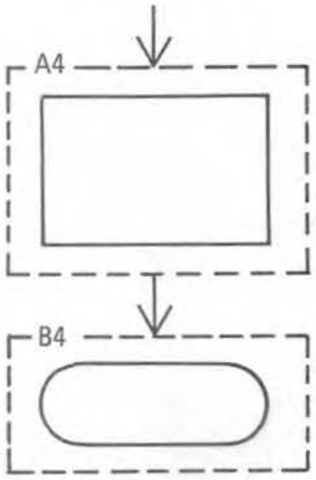
(Strict adherence to this convention is not necessary. Sometimes, as when using the decision box, it would be more convenient to "go against" the flow. If you do, however, make sure that this reverse direction is clearly labeled with arrows and that you get back to the correct flow as soon as possible.)

2. Regardless of whether you are drawing your flowchart on clean paper or on a worksheet, flow lines should connect "symbol to symbol."

PROPER

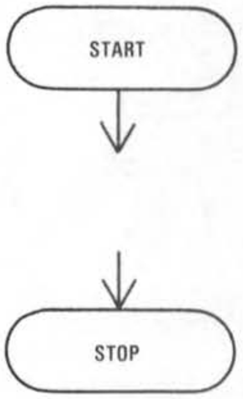


IMPROPER

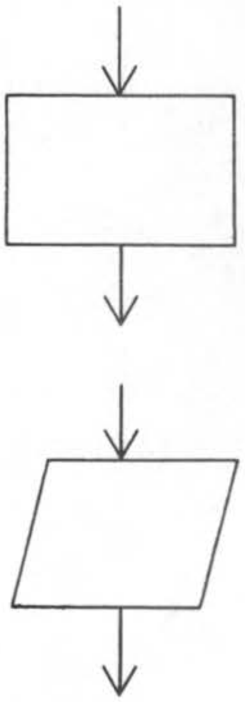


3. Each flowcharting symbol has a limit as to how many flow lines can be connected to it.

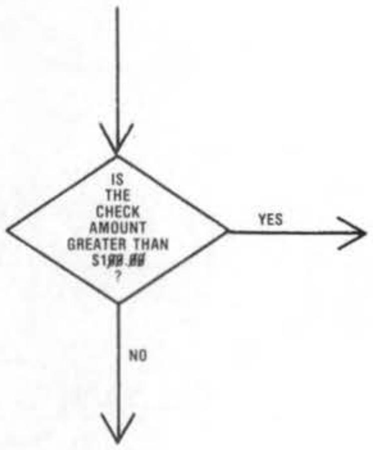
   A. Each terminal symbol can be connected to another symbol by only one flow line. If it is a starting point, it must have one line leading away from it; an ending point has one line leading into it. Examples:

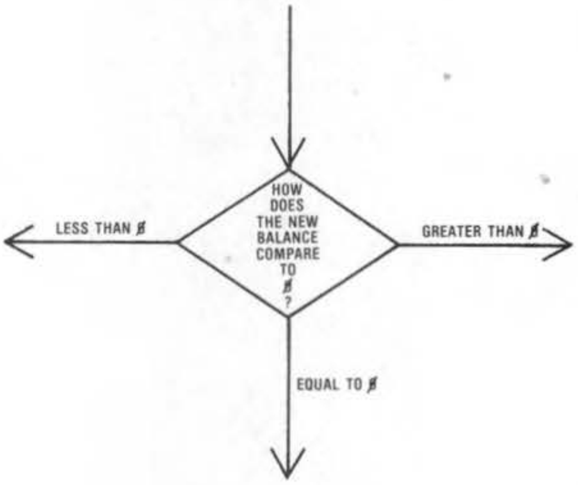


   B. Processing and input/output symbols each have one line leading into them and one line coming out. Examples:

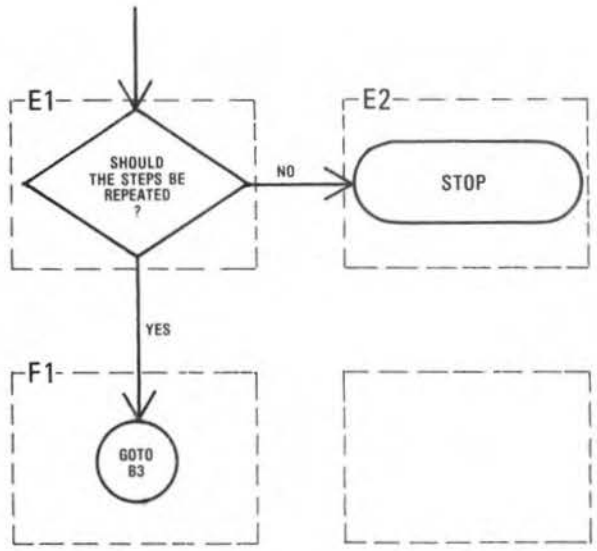


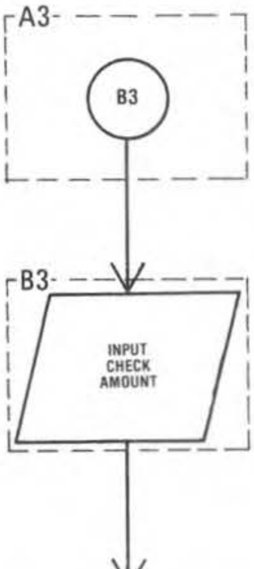
   C. A decision box has one entry line and two or three exit lines, depending on the possible answers to the questions. Examples:





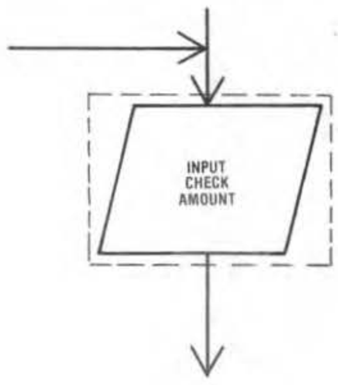
D. Connector symbols have one line leading into them if they are drawn to show where the next instruction box is located. They have one line leading out of them if they are used to highlight where that instruction is. Examples:



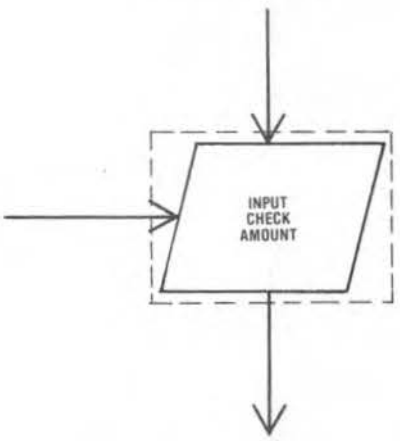


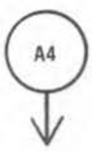
E. If a symbol needs to be entered from more than one point in the program, draw one of the entry lines to the flow line immediately preceding the symbol.

PROPER



IMPROPER





Note that the connector symbol is not always drawn in one of the boxes on the worksheet. It is often useful to draw it in the margins.

That's all we need to know about the generalities of flowcharting. In the next section, we will see some ways in which a flowchart can be used to design specific solutions to computer problems.

First, let's see what we've learned by completing the Programmer's Check below.

PROGRAMMER'S CHECK

1

Flowcharts and Symbols

Let's see how well you can put your knowledge to work. Answer the questions below by doing the drawings and entering the information as required on scratch paper. Then, check your answers. If you have given incorrect answers, review the previous instruction before going further in the lesson.

1. **Draw the symbol and arrow for START and identify it properly.**

2. **Draw a "processing box" and show the steps which will take place within memory as follows: NEW BALANCE = OLD BALANCE — CHECKS + DEPOSITS.**

3. **Draw a box and include appropriate information to show the options for the following: ARE THERE MORE ADDRESSES TO ENTER?**

4. **Draw the connector symbol connected to an input/output symbol to show that "B-2" data is entered as input in the form of a check amount.**

5. **Construct a decision box in which you show three options for the question: HOW DOES THE NEW BALANCE COMPARE TO Ø?**

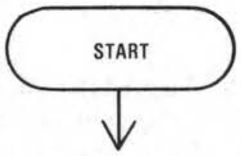
6. **If two entry lines need to enter an input/output symbol, then a certain convention is always used. Can you show how this is done so that the flow is clear?**

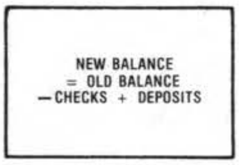
7. **Is it okay for connector symbols to be drawn outside of the boxes on a worksheet or must they always be inside a box?**

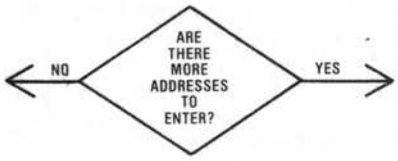
(Answers on Page 12)

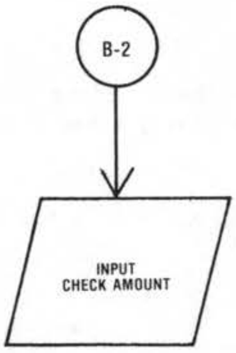
PROGRAMMER'S CHECK ANSWERS

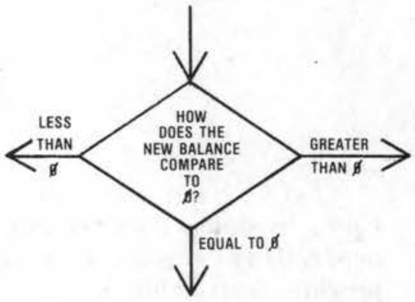
1

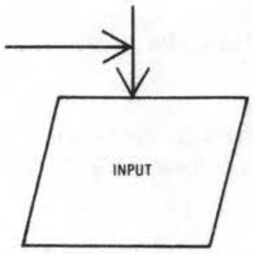
1. 

2. 

3. 

4. 

5. 

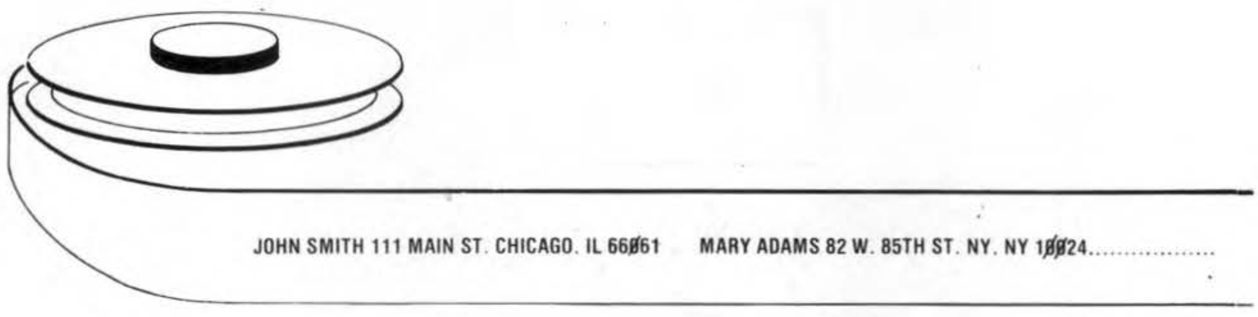
6. 

7. **Connector symbols may be drawn whenever needed. You can use the margin of a worksheet if necessary.**

INPUT/OUTPUT PROGRAMS

Input/output programming is a process by which data is transferred from one medium to another. The data submitted as input (via the keyboard) or read from an auxiliary storage device is held in RAM. A duplicate of the data stored is then printed on paper (or on the CRT).

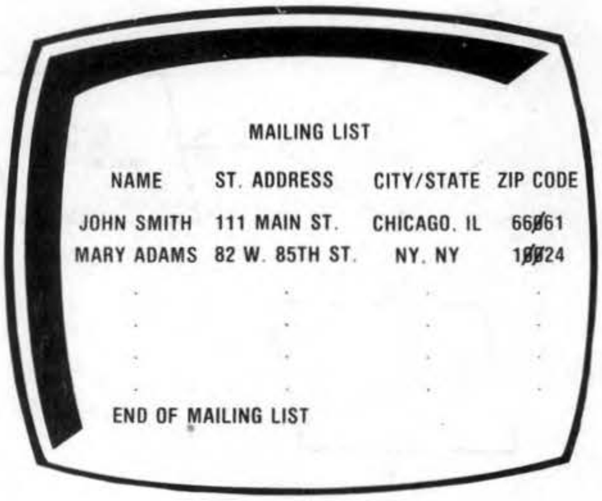
Let's see how input/output can be used in an application program to produce a mailing list from a file stored on tape. On magnetic tape, a file has been previously created. Each record on the file contains a name, a street address, a city, state and zip code. This mailing address file might look something like this:


JOHN SMITH 111 MAIN ST. CHICAGO. IL 66Ø61
MARY ADAMS 82 W. 85TH ST. NY. NY 1ØØ24..................

The format of the input record from this file is defined as having four fields and can be illustrated as:

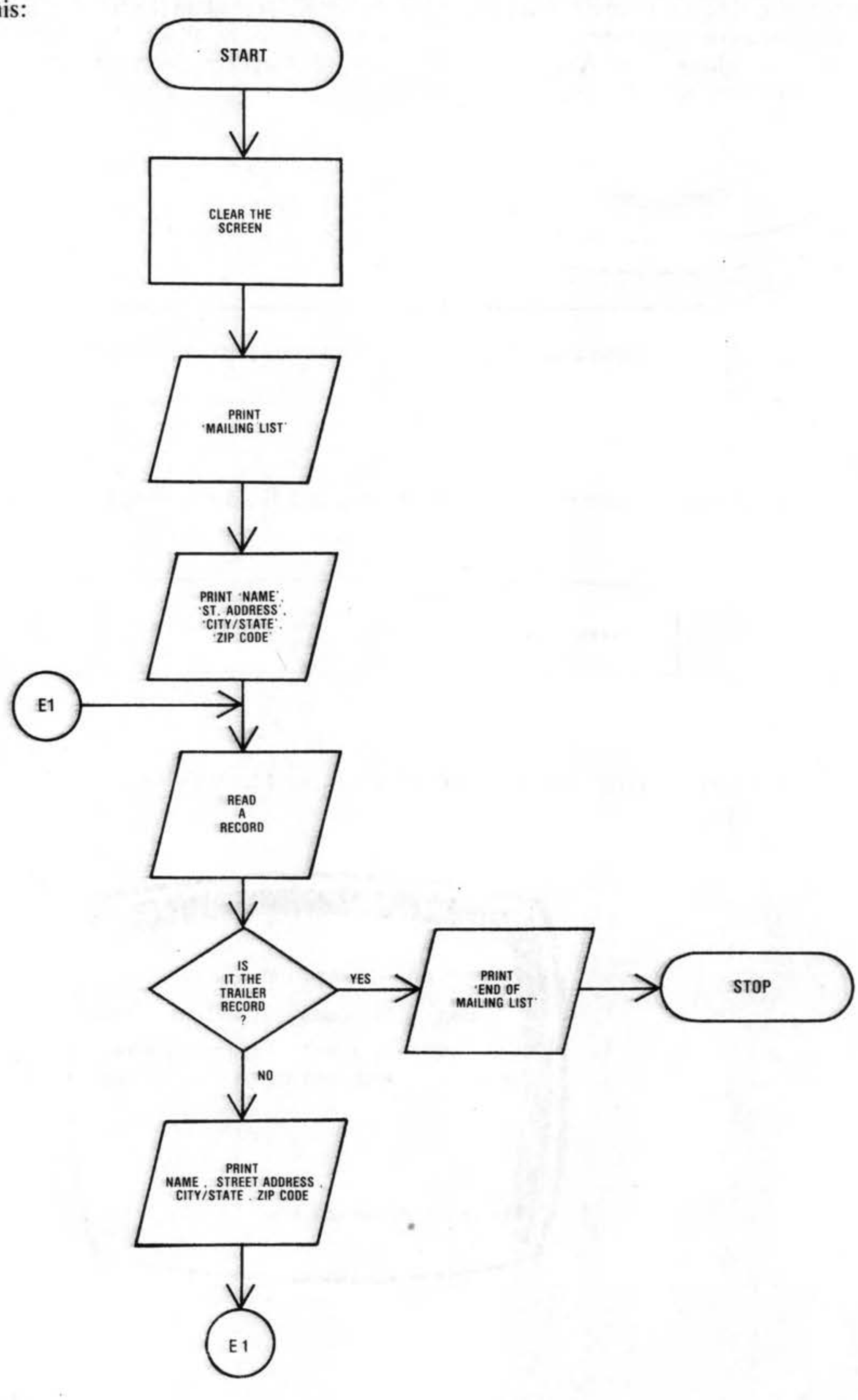
NAME	STREET ADDRESS	CITY/STATE	ZIP CODE

The desired output from this program is a mailing list which is to appear on the screen. It should look like this:



```
              MAILING LIST
   NAME      ST. ADDRESS    CITY/STATE  ZIP CODE
JOHN SMITH  111 MAIN ST.    CHICAGO. IL  66Ø61
MARY ADAMS  82 W. 85TH ST.    NY. NY     1ØØ24
    .            .              .          .
    .            .              .          .
    .            .              .          .
    .            .              .          .
   END OF MAILING LIST
```

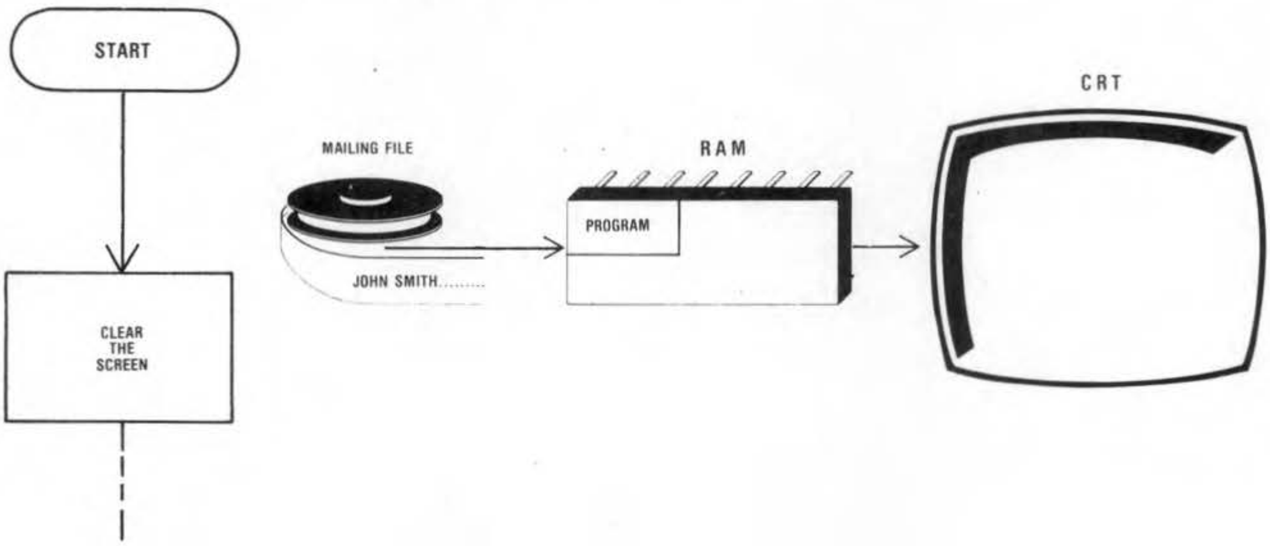
The program design (flowchart) which will produce the correct output from the given input would look like this:



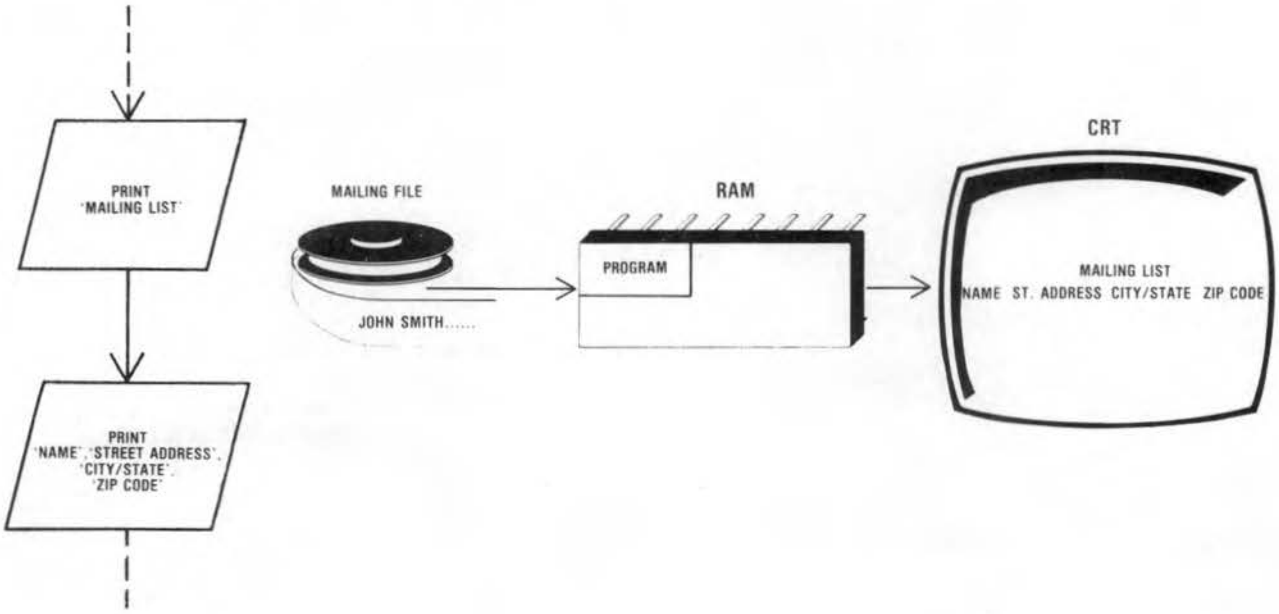
START
CLEAR THE SCREEN
PRINT 'MAILING LIST'
PRINT 'NAME', 'ST. ADDRESS', 'CITY/STATE', 'ZIP CODE'
E1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
PRINT 'END OF MAILING LIST'
STOP
NO
PRINT NAME , STREET ADDRESS , CITY/STATE , ZIP CODE
E1

Let's examine this flowchart in detail to see how this program should work when it is commanded to run.

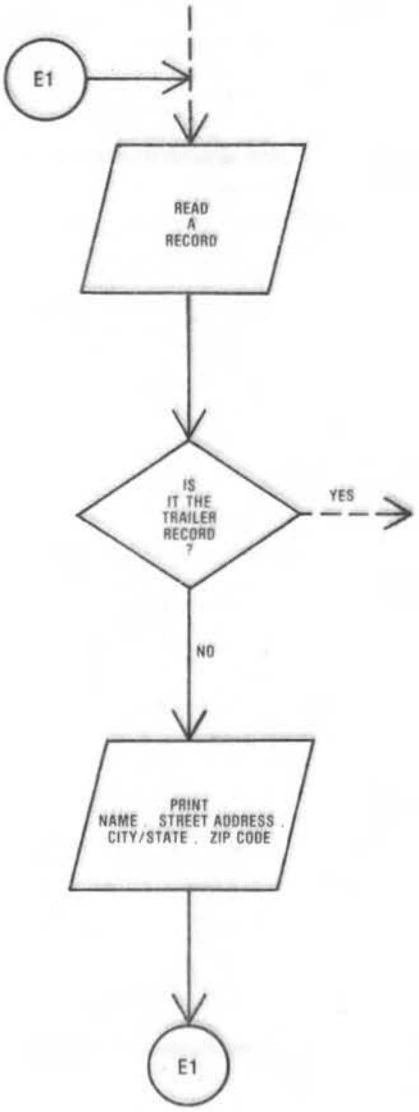
We begin with our starting point, the terminal symbol. Our first instruction is a command to clear the screen of any characters which were previously left on it. After execution of this instruction, the system looks like this:



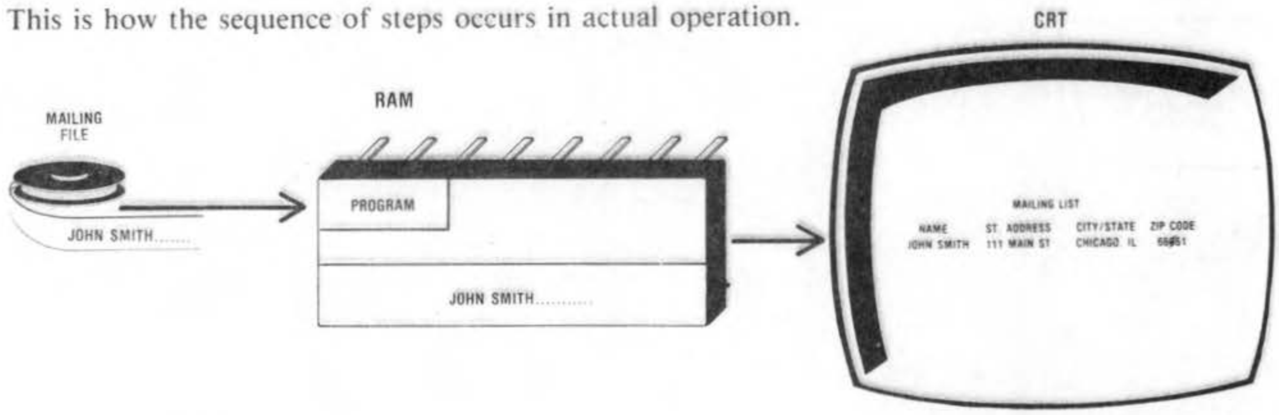
Our next two statements are instructions to cause the information to be printed on the screen: a title for the report and column headings to indicate the meaning of the data to be printed underneath them. Note that, in the flowchart symbols, the words to be printed are enclosed within single quotation marks. This signifies that the words are to be written as "constant" data—that is, exactly as shown. After these statements are entered, this is what we would see:


PRINT 'MAILING LIST'
PRINT 'NAME','STREET ADDRESS','CITY/STATE','ZIP CODE'
MAILING FILE
JOHN SMITH......
RAM
PROGRAM
CRT
MAILING LIST
NAME ST. ADDRESS CITY/STATE ZIP CODE

We now begin our loop—that is, the sequence of steps which will be repeated for each record on the file.


E1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
NO
PRINT NAME , STREET ADDRESS , CITY/STATE , ZIP CODE
E1

This is how the sequence of steps occurs in actual operation.

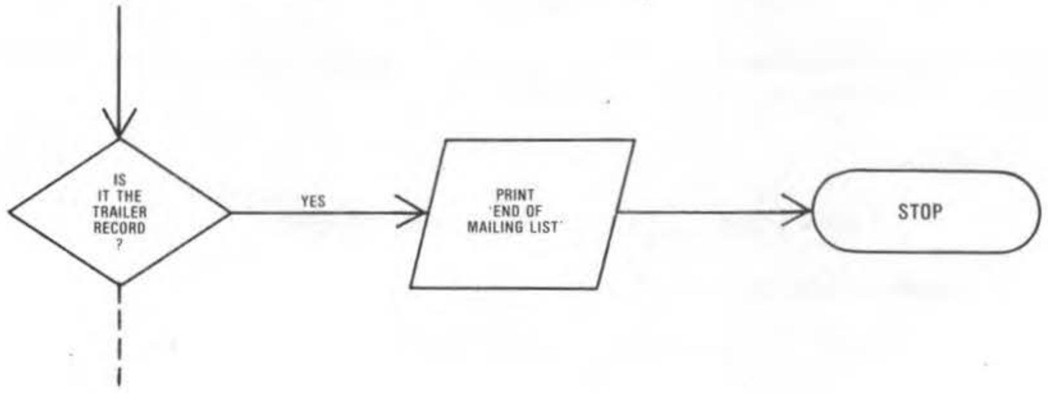


We first see a connector symbol, $\textcircled{E1}\rightarrow$. Following the connector symbol is the first instruction within the loop, a command to read a record from our mailing address file. Since we don't know how many records will be on the file at the time we design and code our program, we must test each record read to see if it is the "trailer record." The trailer record is placed at the end of a file to indicate that all records have been previously accessed.

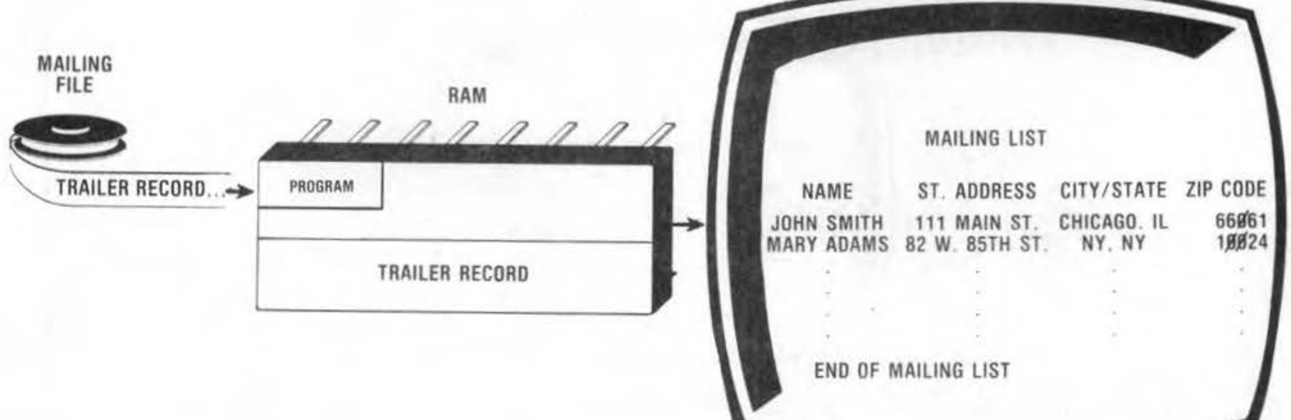
If the record read is not the trailer record, then a line will be printed on the screen, with the name, street address, city/state and zip code on it. Following this, our program branches or loops back to read the next record from the file.

This process will be repeated until a line is printed on the CRT for each record that is read, regardless of how many records are on the file. (Remember, however, that there is a limit to the number of lines of text that the screen can hold at one time.)

Eventually, the trailer record will be read. This record will cause the "yes" path out of the decision box to be taken, the sequence of steps leading to the end of our program.



A final line is printed on the screen, showing that the program is over. Finally, our program relinquishes control of the computer system so that other programs can be run.



```
                    MAILING LIST

NAME         ST. ADDRESS     CITY/STATE    ZIP CODE
JOHN SMITH   111 MAIN ST.    CHICAGO. IL      66Ø61
MARY ADAMS   82 W. 85TH ST.   NY, NY          1ØØ24
    .             .              .               .
    .             .              .               .
    .             .              .               .

END OF MAILING LIST
```

The mailing list input/output program is our first attempt at controlling the computer's enormous potential. We will soon be learning more advanced techniques involving arithmetic and comparing. But before we move on, let's see what we've learned by completing the Programmer's Check below.

PROGRAMMER'S CHECK

2

Flowcharting

Write a flowchart to produce a listing of telephone numbers from records on a file. Use a separate sheet of paper. The file looks like this:

RECORD LAYOUT:

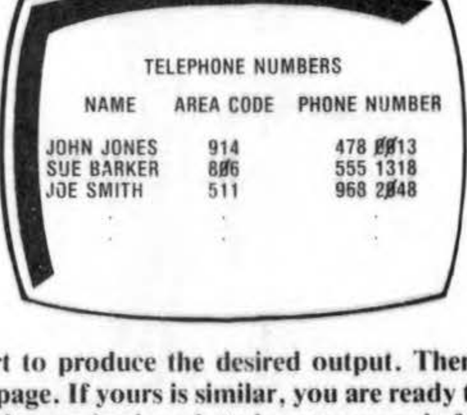
AREA CODE PHONE NUMBER NAME

Some of the records on this file look like this:

```
914   478  ØØ13   JOHN JONES
8Ø6   555  1318   SUE BARKER
511   968  2Ø48   JOE SMITH
 .     .     .     .
 .     .     .     .
```

The output, which will be on the screen, will have headings and a final line in addition to the name, area code and phone number from each record.

OUTPUT:



```
              TELEPHONE NUMBERS
      NAME     AREA CODE   PHONE NUMBER

JOHN JONES        914         478 ØØ13
SUE BARKER        8Ø6         555 1318
JOE SMITH         511         968 2Ø48
     .             .              .
     .             .              .
```

Now, code the flowchart to produce the desired output. Then, compare your flowchart with the one on the indicated page. If yours is similar, you are ready to go on to the next section. If your chart is not identical, then go back and study some more before attempting to flowchart the above file again. Good luck!

(Answers on Page 20)

ARITHMETIC

Now let's add some calculations to our program applications. This will give us the ability to print more information than we read from our input—that is, the computer will produce new information in the form of results.

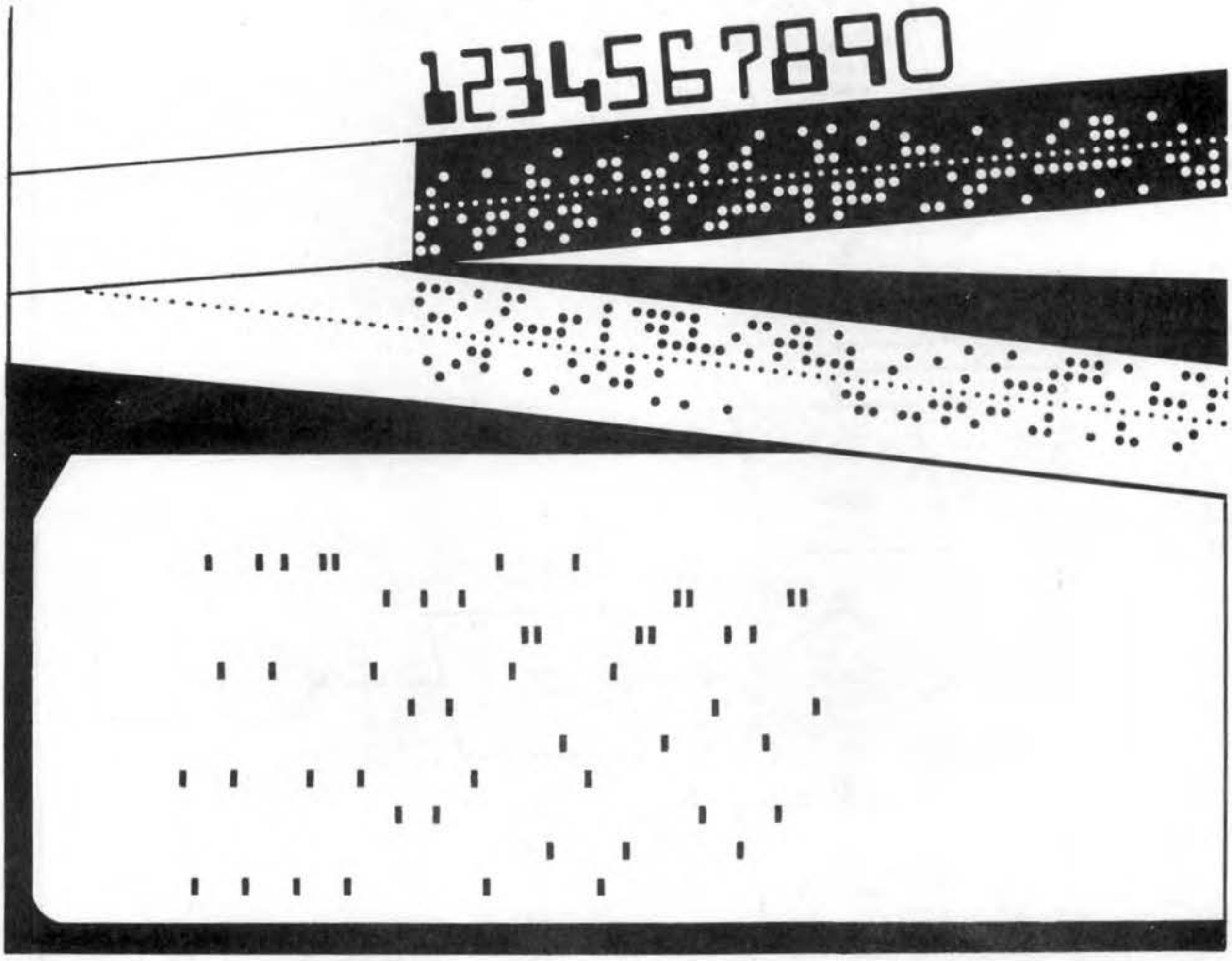
Our computer can easily perform all of the basic arithmetic operations: addition, subtraction, multiplication, division and exponentiation (raising a number to a power). With these capabilities, we not only can do calculations on each record, we can also accumulate totals of all records on the file and even determine overall averages.

Arithmetic statements are entered on a flowchart as processing symbols. In most programming languages (BASIC included), more than one arithmetic operation can be performed in one statement, as in:

NEW BALANCE = OLD BALANCE + DEPOSITS — WITHDRAWALS

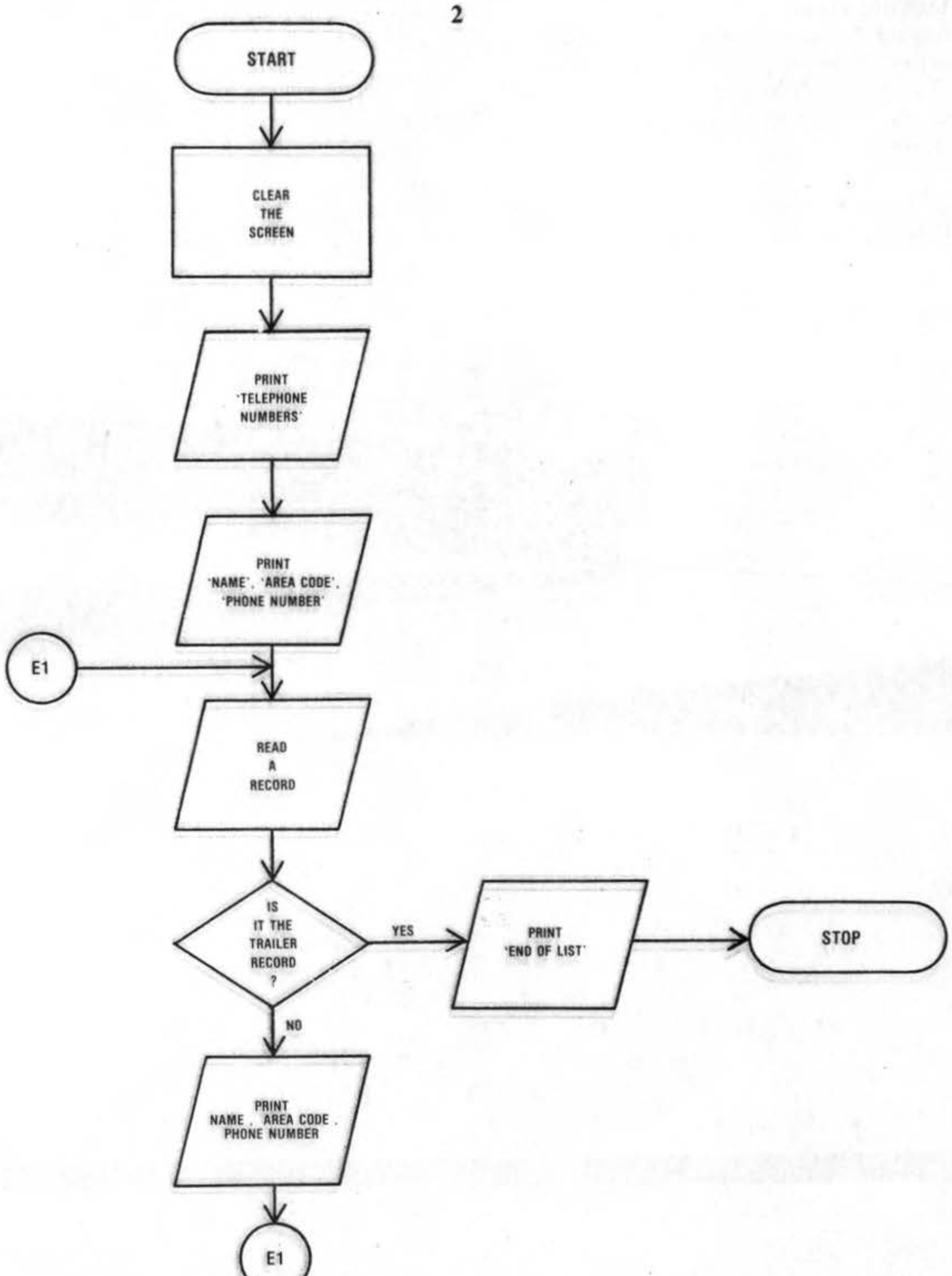
When a program is run, values are substituted for the variables on the right of the equals sign (=) and the results are placed in the variable to the left.

An example of a flowchart which adds arithmetic to its I/O operations is shown on page 21. Notice how calculations for each record are done within the loop, including the accumulating of final totals which will be printed only towards the end of the program.



PROGRAMMER'S CHECK ANSWERS

2

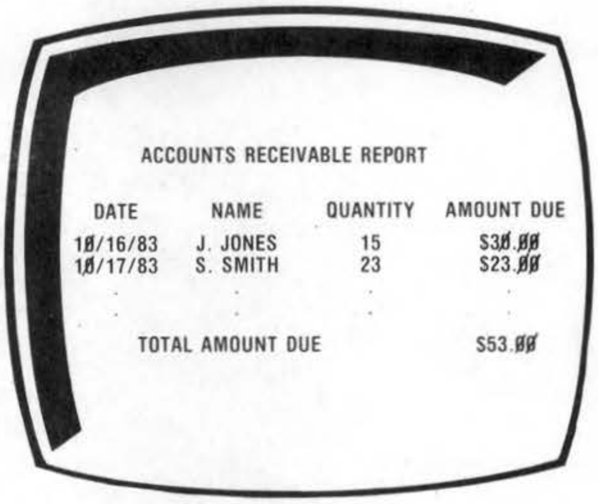

START
CLEAR THE SCREEN
PRINT 'TELEPHONE NUMBERS'
PRINT 'NAME', 'AREA CODE', 'PHONE NUMBER'
E1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
PRINT 'END OF LIST'
STOP
NO
PRINT NAME , AREA CODE , PHONE NUMBER
E1

ACCOUNTS RECEIVABLE REPORT APPLICATION

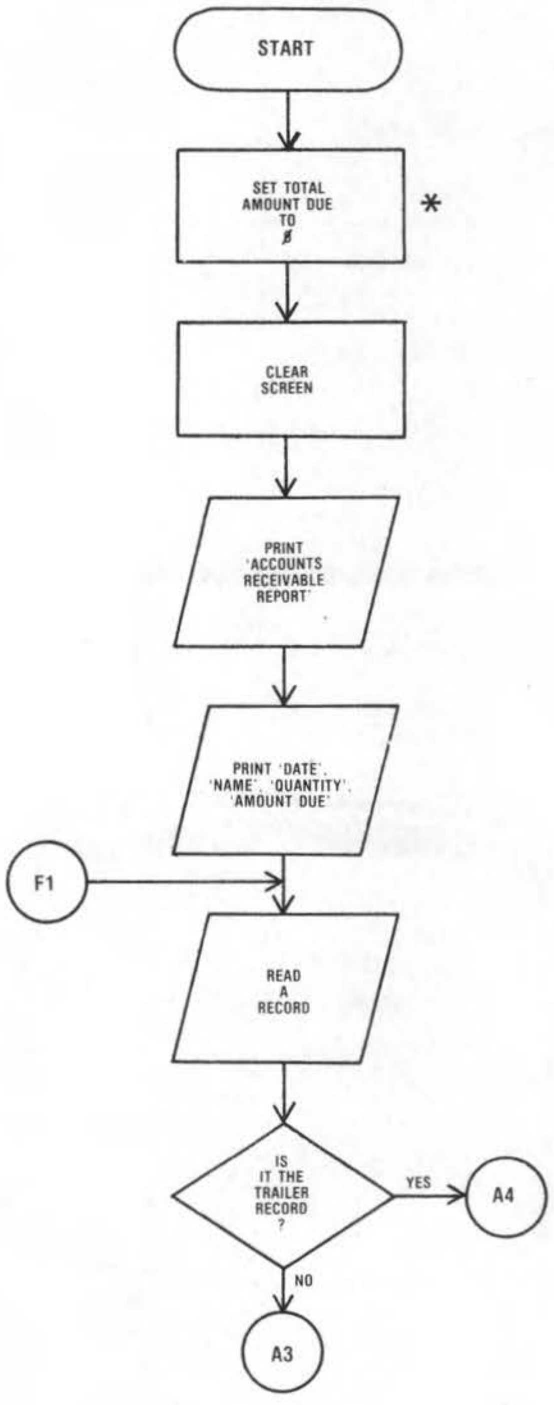
INPUT: ACCOUNTS RECEIVABLE FILE

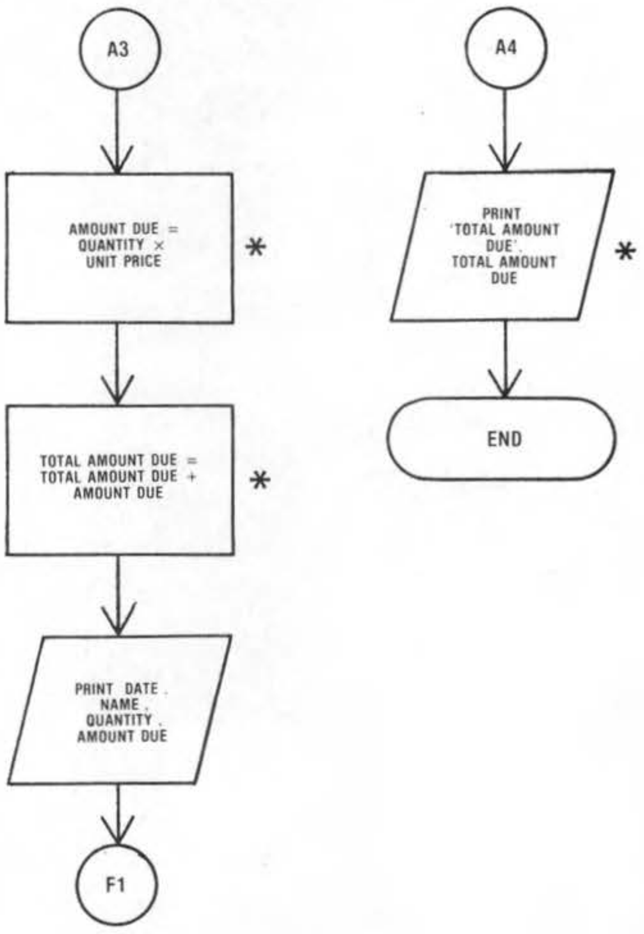
DATE	NAME	QUANTITY	UNIT PRICE
1Ø/16/83	J. JONES	15	$2.ØØ
1Ø/17/83	S. SMITH	23	$1.ØØ

OUTPUT: ACCOUNTS RECEIVABLE REPORT

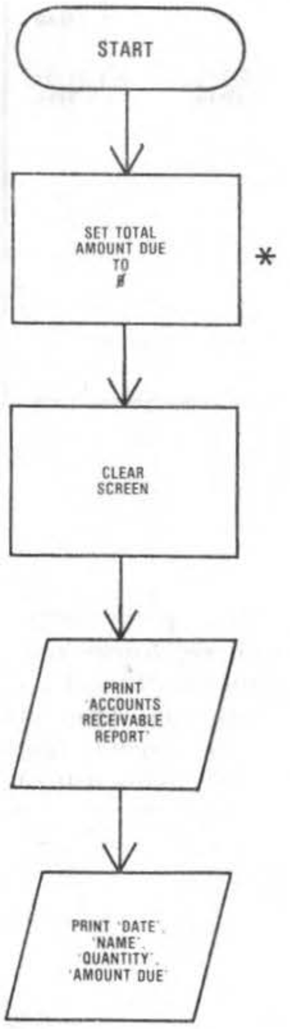


```
ACCOUNTS RECEIVABLE REPORT

DATE       NAME       QUANTITY   AMOUNT DUE
1Ø/16/83   J. JONES      15        $3Ø.ØØ
1Ø/17/83   S. SMITH      23        $23.ØØ
   .          .           .           .
   .          .           .           .

TOTAL AMOUNT DUE                   $53.ØØ
```

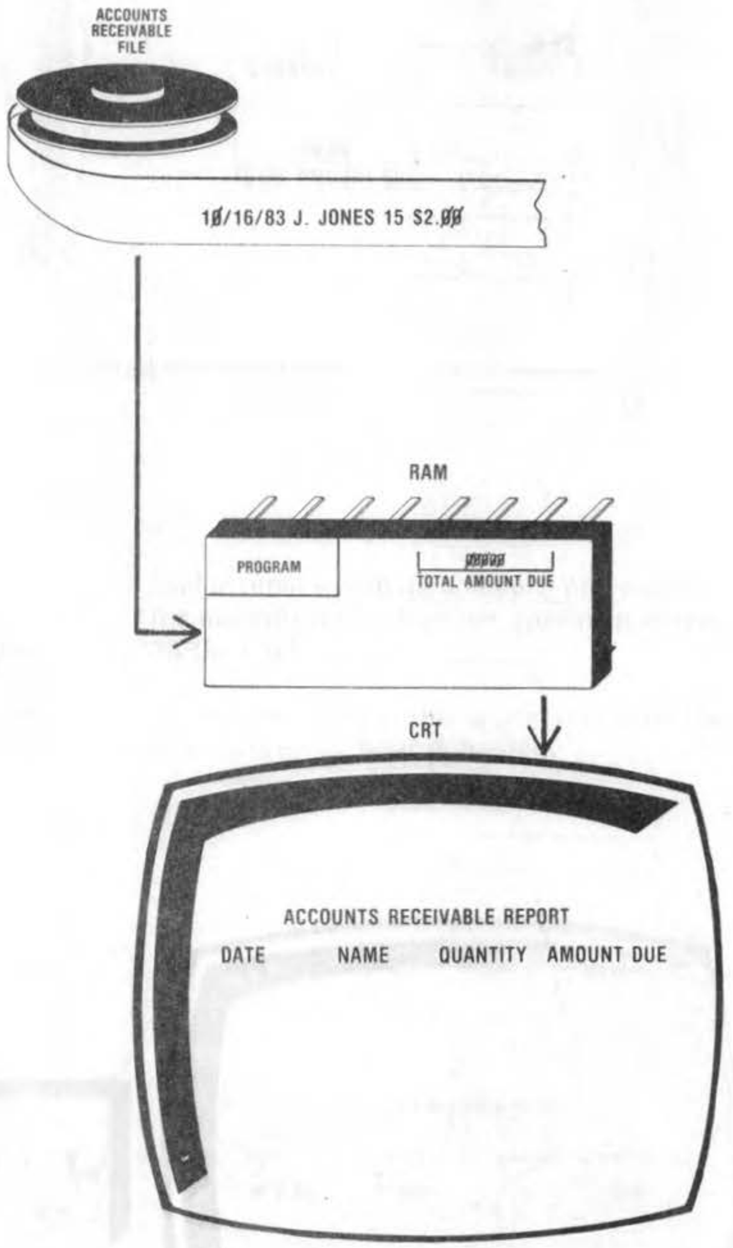

START
SET TOTAL AMOUNT DUE TO Ø
*
CLEAR SCREEN
PRINT 'ACCOUNTS RECEIVABLE REPORT'
PRINT 'DATE', 'NAME', 'QUANTITY', 'AMOUNT DUE'
F1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
A4
NO
A3


A3
AMOUNT DUE = QUANTITY × UNIT PRICE
*
TOTAL AMOUNT DUE = TOTAL AMOUNT DUE + AMOUNT DUE
*
PRINT DATE, NAME, QUANTITY, AMOUNT DUE
F1
A4
PRINT 'TOTAL AMOUNT DUE', TOTAL AMOUNT DUE
*
END

Let's walk through this flowchart, paying special attention to the starred (*) symbols. In our "initialization routine" shown below, our memory and peripherals look like this:



START
SET TOTAL AMOUNT DUE TO Ø
*
CLEAR SCREEN
PRINT 'ACCOUNTS RECEIVABLE REPORT'
PRINT 'DATE', 'NAME', 'QUANTITY', 'AMOUNT DUE'

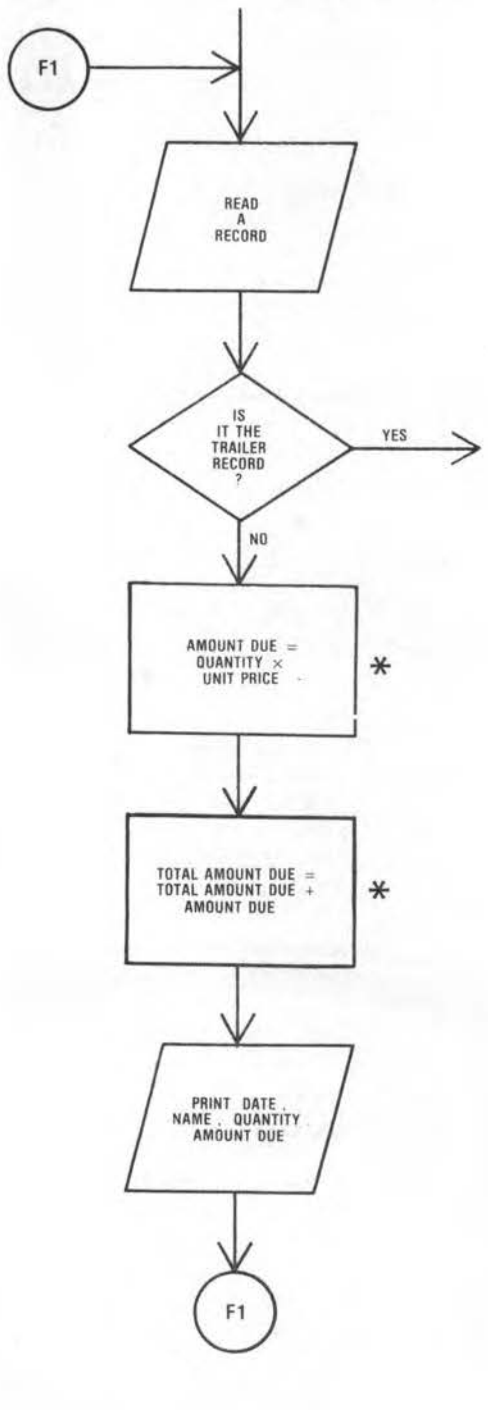


ACCOUNTS RECEIVABLE FILE
1Ø/16/83 J. JONES 15 $2.ØØ
RAM
PROGRAM
ØØØØ
TOTAL AMOUNT DUE
CRT
ACCOUNTS RECEIVABLE REPORT
DATE NAME QUANTITY AMOUNT DUE

Because our report requires a final total of all the individual amounts due, we will have to add them together within our program. To accomplish this, we establish a variable in main storage to serve as "counter." In our loop, we will see how this will work. But first, it is essential that we begin our counter at Ø so our results will be valid.

We have already seen how the next three statements work: 1) the screen is cleared, 2) the title line is printed at the top of the screen, and 3) the column headings are printed. (Remember, single quotation marks are placed around words when they are to be printed exactly as they are written.)

Now, we enter our program loop:


F1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
NO
AMOUNT DUE = QUANTITY × UNIT PRICE
*
TOTAL AMOUNT DUE = TOTAL AMOUNT DUE + AMOUNT DUE
*
PRINT DATE, NAME, QUANTITY AMOUNT DUE
F1

INITIALIZATION ROUTINE

- **Clearing the screen**
- **Printing the headings**
- **Setting counters and input locations**

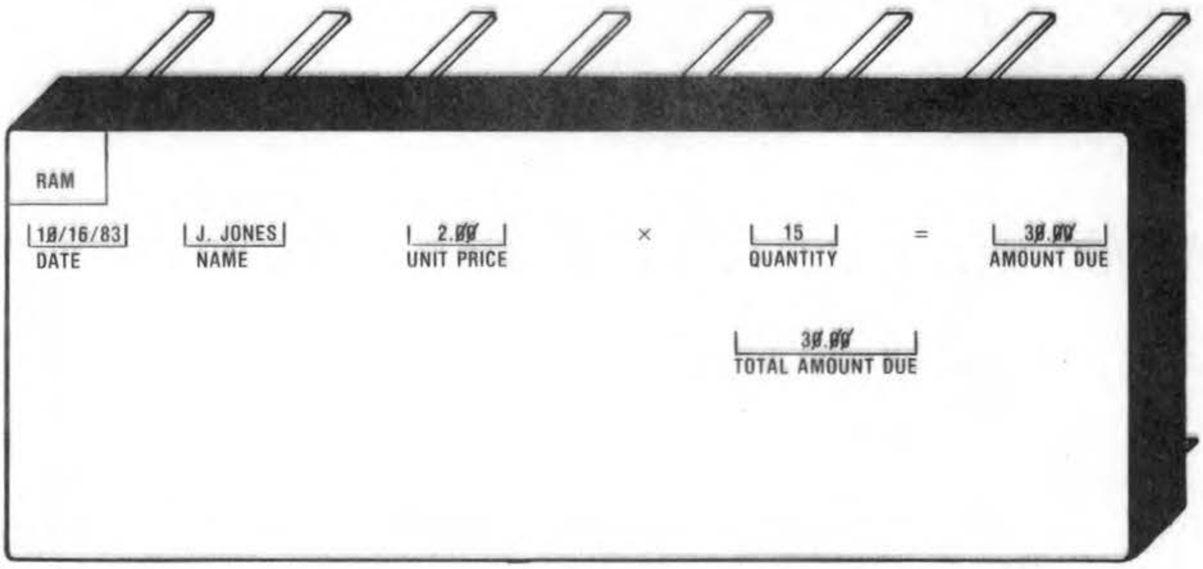
FIGURE 5—There are routines you set up at the outset so the computer can receive data and process it accurately. These will become automatic.

As in our previous programs, a record is read. In this case, the record consists of four fields: date, name, quantity and unit price. As long as this is not the trailer record, the loop is begun.

Our first arithmetic processing box instructs the computer to use the actual values just read for the quantity and unit price, to multiply them together, and to store the results under the variable name, amount due. This variable, while not on the input record, is used in our next processing box at the top of page 24. It will take the value just calculated as the amount due and add it to the counter, total amount due, and will store the new total amount due.

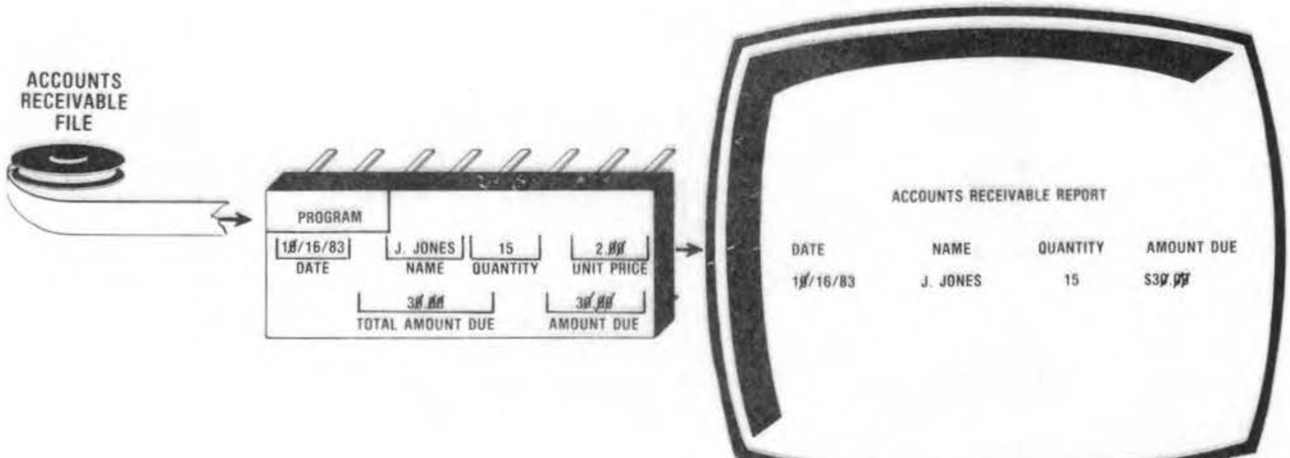
If our first record had a unit price value of 2.ØØ and a quantity of 15, our instruction would result in 3Ø.ØØ being stored as the amount due.

In the second arithmetic processing box, the 3Ø.ØØ just calculated is added to the total amount due counter which had just been set to Ø.



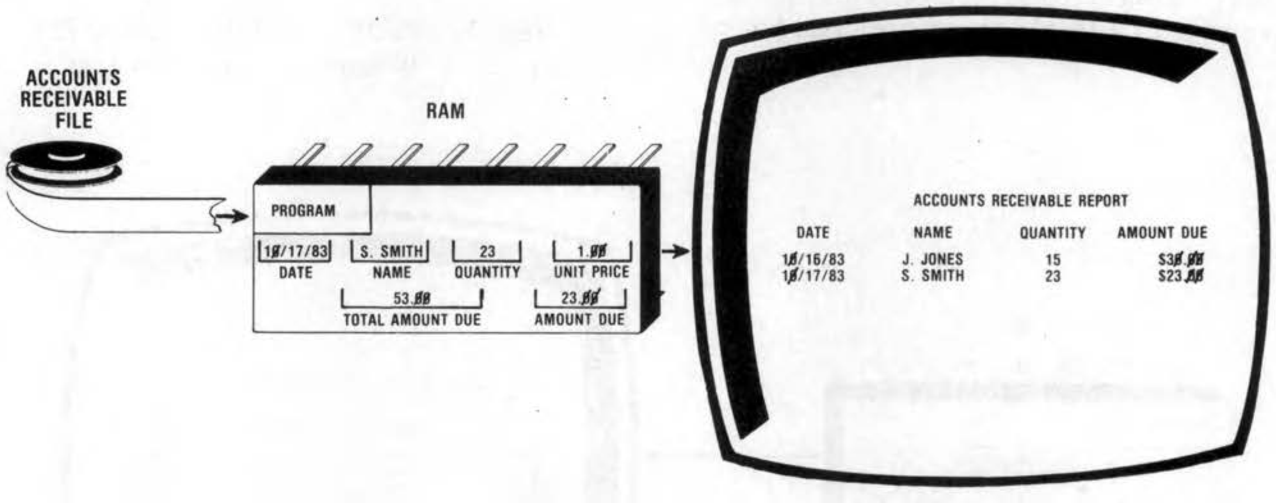
Having done all the calculations necessary for this record, we instruct the computer to write a line of output on the CRT, consisting of the order date, the customer's name, the quantity of items ordered, and the amount due. Notice that although the unit price variable was read as input within the computer and was used for multiplying, it does not appear in output on the CRT.

Our system has this appearance after the first record has been processed:



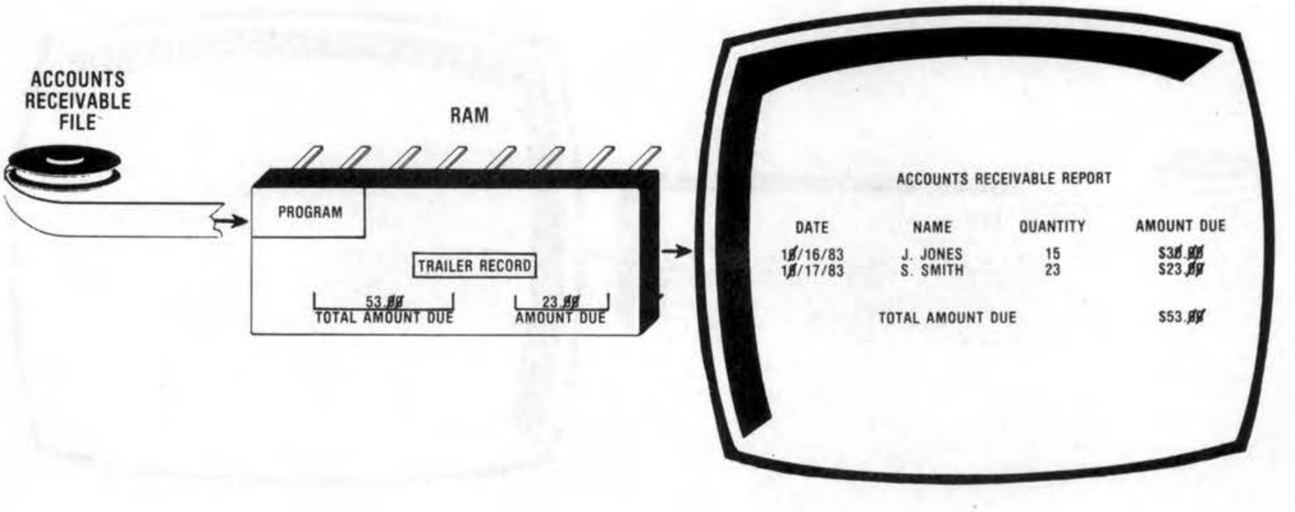
Now, let's follow the path of the second record as we connect back to the "read" statement.

The second record, as long as it's not the trailer record, will also have its amount due calculated. This result will be added to the 3Ø.ØØ already stored in total amount due, then it will be written to the screen.



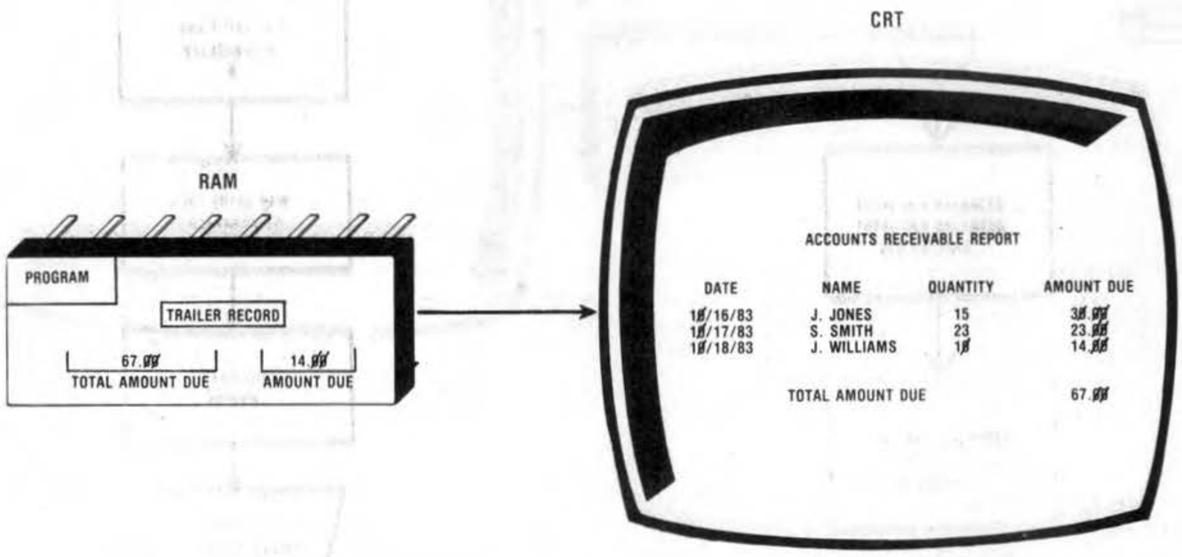
This processing will be repeated until our trailer record is encountered. Let us imagine for this example that the trailer record is found on the third "read."

The "yes" path will be taken from our decision box, commanding the computer to display a line of output consisting of the words "TOTAL AMOUNT DUE," followed by the value in total amount due variable, added in our loop. The system then ends up looking like this:



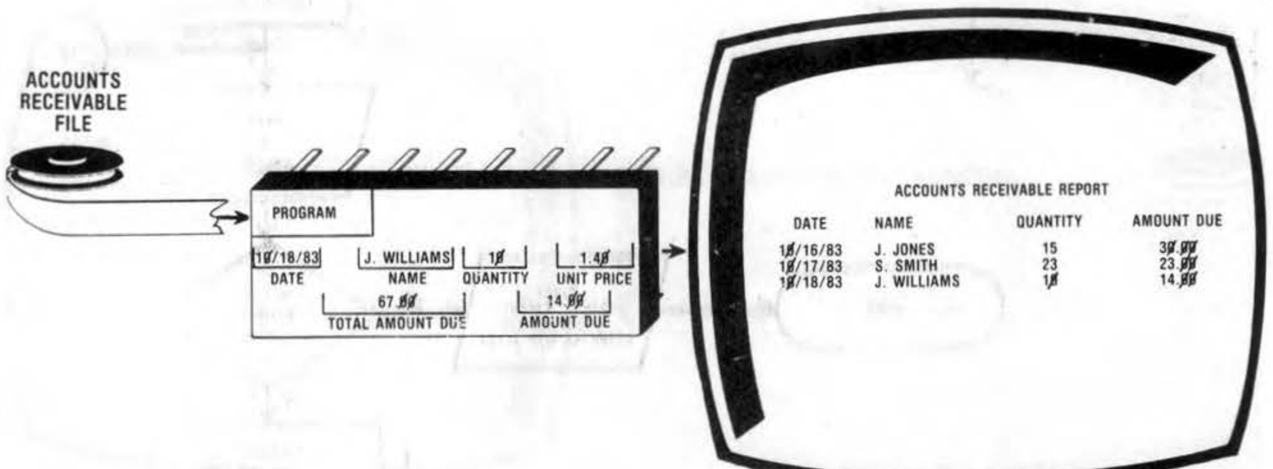
In order to prove that this program design will work, regardless of how many records are read, we would repeat the process demonstrated on page 25 with one or more records. For example, let's input the following additional record:

Date	Name	Quantity	Unit Price
1Ø/18/83	J. Williams	1Ø	1.4Ø



```
                ACCOUNTS RECEIVABLE REPORT

DATE        NAME          QUANTITY        AMOUNT DUE
1Ø/16/83    J. JONES         15              3Ø.ØØ
1Ø/17/83    S. SMITH         23              23.ØØ
1Ø/18/83    J. WILLIAMS      1Ø              14.ØØ

           TOTAL AMOUNT DUE                  67.ØØ
```

Once all additional records are entered, the "trailer" record would again end the loop. The trailer record would produce the total amount due on the CRT.



```
                ACCOUNTS RECEIVABLE REPORT
DATE        NAME              QUANTITY        AMOUNT DUE
1Ø/16/83    J. JONES             15              3Ø.ØØ
1Ø/17/83    S. SMITH             23              23.ØØ
1Ø/18/83    J. WILLIAMS          1Ø              14.ØØ
```

As soon as you feel comfortable with the applications just illustrated, do the Programmer's Check which follows to see how much you have learned.

PROGRAMMERS CHECK

3

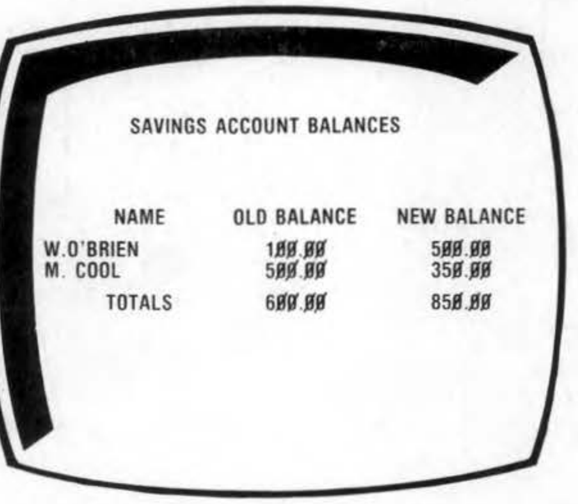
Flowcharting Bank Accounts

The object of this flowchart is to design a program which will print the results of transactions made to various bank savings accounts and to print final totals. Do your flowcharting on a separate sheet of paper.

INPUT: ACCOUNT FILE

NAME	ACCOUNT NUMBER	OLD BALANCE	DEPOSITS	WITHDRAWALS
W. O'Brien	1Ø8368	1ØØ.ØØ	6ØØ.ØØ	2ØØ.ØØ
M. Cool	85113Ø	5ØØ.ØØ	15Ø.ØØ	3ØØ.ØØ
TRAILER RECORD				

OUTPUT: CRT



```
             SAVINGS ACCOUNT BALANCES


          NAME        OLD BALANCE    NEW BALANCE
W.O'BRIEN               1ØØ.ØØ          5ØØ.ØØ
M. COOL                 5ØØ.ØØ          35Ø.ØØ
          TOTALS        6ØØ.ØØ          85Ø.ØØ
```

PROCESSING:

The calculations necessary for this program are as follows:

NEW BALANCE = OLD BALANCE + DEPOSITS — WITHDRAWALS

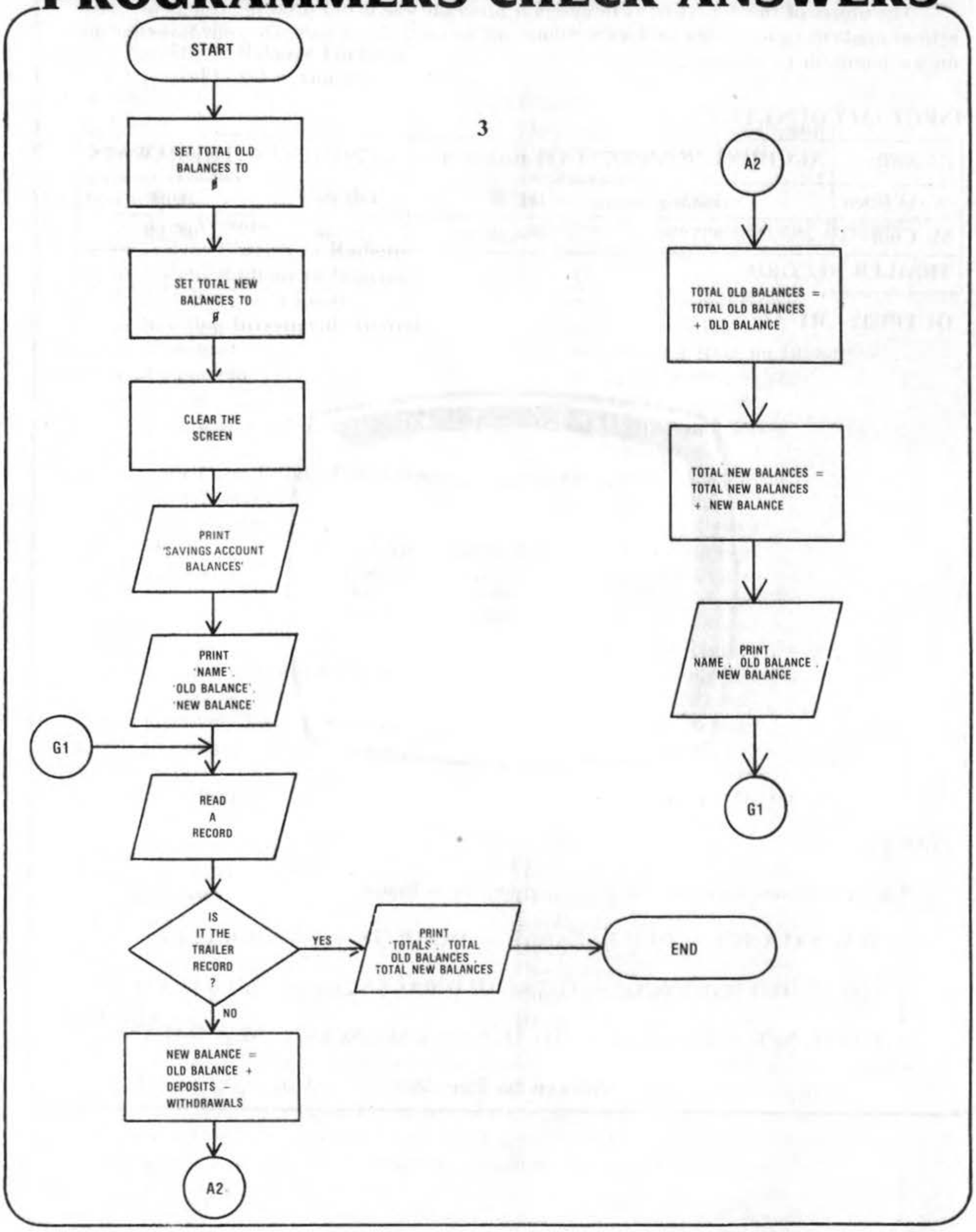
TOTAL OLD BALANCES = TOTAL OLD BALANCES + OLD BALANCE

TOTAL NEW BALANCES = TOTAL NEW BALANCES + NEW BALANCE

(Answers on Page 28)

PROGRAMMERS CHECK ANSWERS

3

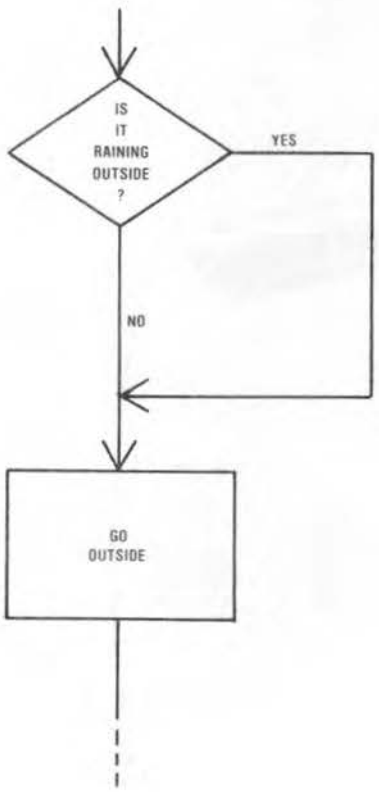

START
SET TOTAL OLD BALANCES TO Ø
SET TOTAL NEW BALANCES TO Ø
CLEAR THE SCREEN
PRINT 'SAVINGS ACCOUNT BALANCES'
PRINT 'NAME', 'OLD BALANCE', 'NEW BALANCE'
G1
READ A RECORD
IS IT THE TRAILER RECORD ?
YES
PRINT 'TOTALS', TOTAL OLD BALANCES, TOTAL NEW BALANCES
END
NO
NEW BALANCE = OLD BALANCE + DEPOSITS − WITHDRAWALS
A2
A2
TOTAL OLD BALANCES = TOTAL OLD BALANCES + OLD BALANCE
TOTAL NEW BALANCES = TOTAL NEW BALANCES + NEW BALANCE
PRINT NAME, OLD BALANCE, NEW BALANCE
G1

Keep in mind that the computer can do only three basic operations: input/output, arithmetic, and comparing. We have just seen how we can design programs which use the first two capabilities. Now, let's look at the third, logic.

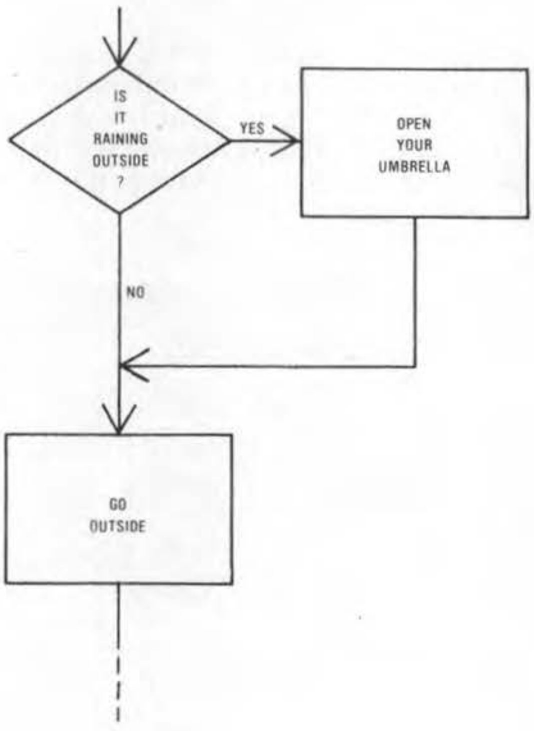
Computer logic involves the comparing of two values by the CPU. As a result of the comparison, one of three conditions will be true: the values will be equal (=), the first value will be greater than (>) the second value, or the first will be less than (<) the second.

Comparing involves the asking of a question. A flowchart uses the diamond-shaped decision box to illustrate this. But asking a question ought to imply listening to the answer—that is, taking alternative actions depending on the answer to the question. In computer programming, we refer to this as "branching."

Although the following is possible, think of how ridiculous it appears:



Regardless of the answer to the question, the same action is taken. This type of question should never be posed in a program unless *alternative* actions are taken. Now, contrast the first example with the one below:



Note that, this time, something special (the opening of your umbrella) will take place as a result of the answer to the question. This is the way in which we give our programs the flexibility to process data differently according to our needs.

Even in our first input/output flowchart, we used computer logic. Do you recall where? Remember, whenever we establish a loop, we must provide a way out of the loop. As we read each record, we checked to see if it was the trailer record so that we could direct the program towards the end. This was a comparison, or question, and as a result we were able to take alternative actions based upon the answer—either to continue the loop if the record read was not the trailer record, or to terminate the loop if it was.

Let's take a look at another more involved application which uses computer logic: an employee weekly payroll program.

Our employee payroll file will serve as input to our program. It consists of records containing four fields: an employee number, an employee name, the number of hours worked for the week, and the normal, hourly pay rate. This is illustrated below:

EMPLOYEE PAYROLL RECORD

EMPLOYEE NUMBER	EMPLOYEE NAME	HOURS WORKED	PAY RATE

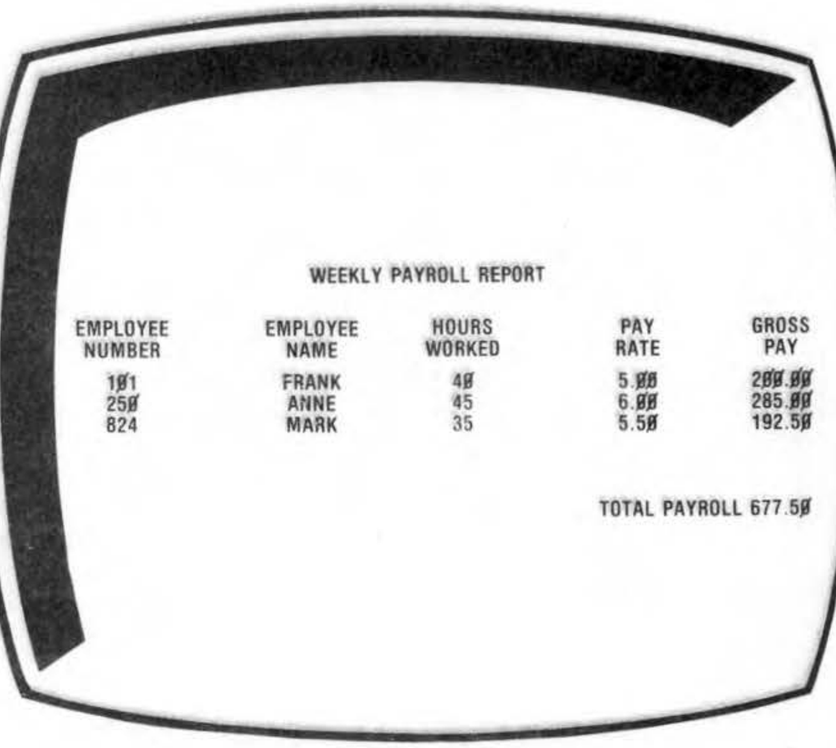
Our output should consist of detail lines which contain, in addition to the four input fields, the employee's gross pay. But, in order to calculate the gross pay, we will have to determine whether or not the employee is entitled to overtime pay. Overtime pay will be given for all hours worked after the first 4Ø hours at time-and-a-half the regular pay rate. Also, a final total of all employees' gross pays will be taken.

Based on the following input data, at the conclusion of our program, we would see this output:

INPUT FILE

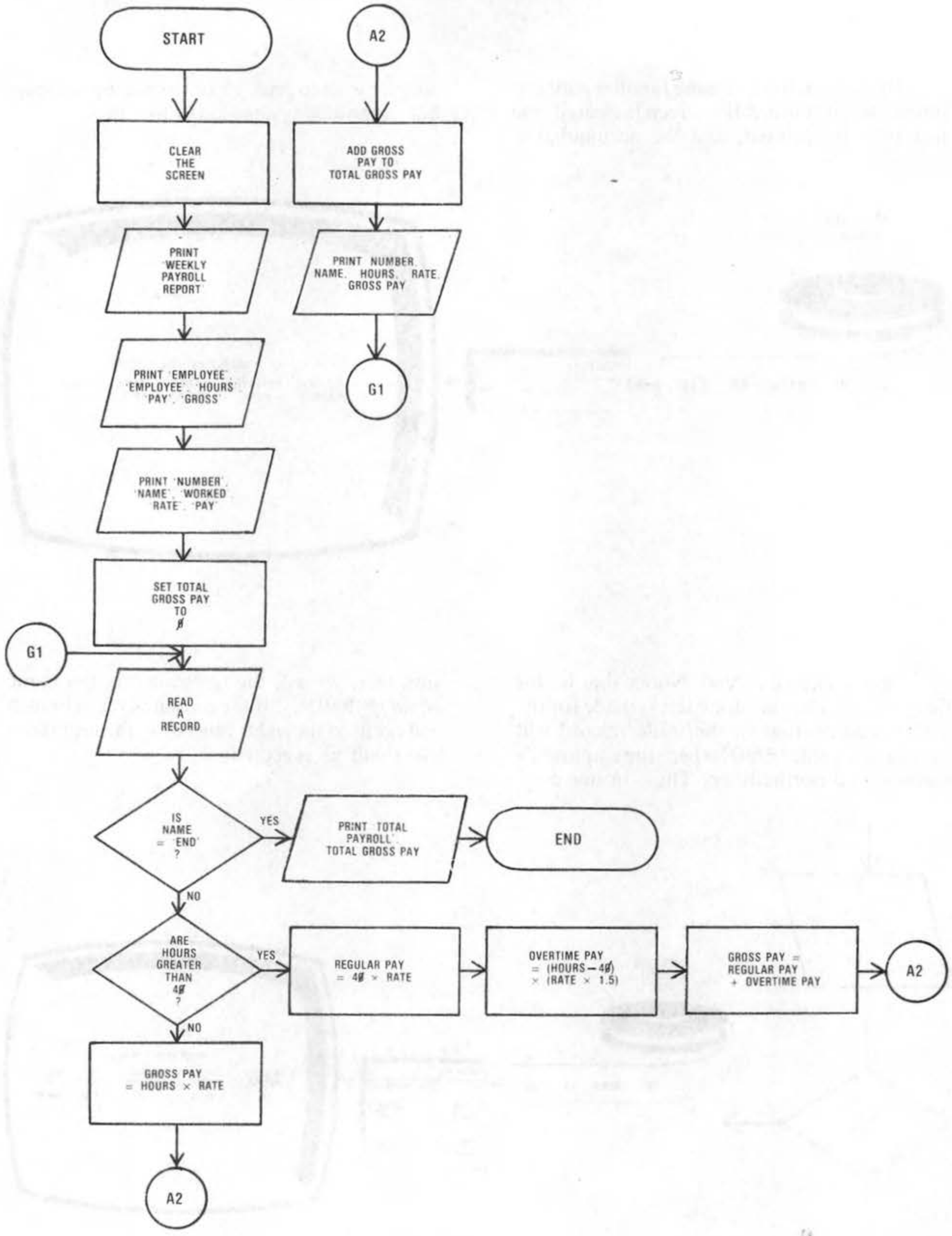
Emp.#	Name	Hours	Pay Rate/ Hour
1Ø1	Frank	4Ø	5.ØØ
25Ø	Anne	45	6.ØØ
824	Mark	35	5.5Ø
999	End	99	9.99

CRT OUTPUT:



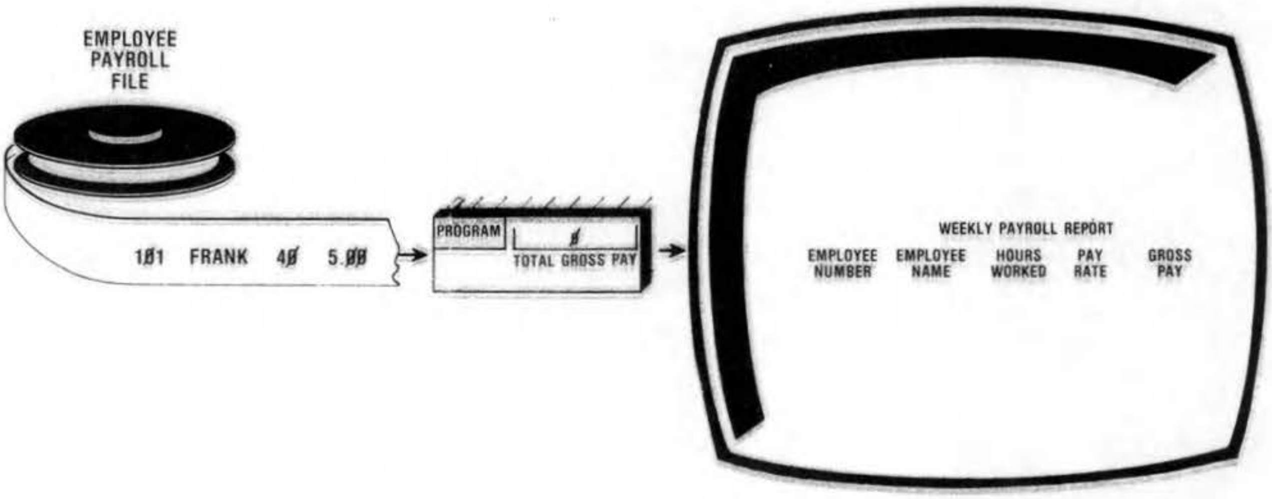
```
                      WEEKLY PAYROLL REPORT

EMPLOYEE      EMPLOYEE      HOURS        PAY       GROSS
 NUMBER         NAME        WORKED       RATE       PAY
  1Ø1          FRANK         4Ø          5.ØØ      2ØØ.ØØ
  25Ø          ANNE          45          6.ØØ      285.ØØ
  824          MARK          35          5.5Ø      192.5Ø

                                   TOTAL PAYROLL 677.5Ø
```

The flowchart design to solve this problem appears below:



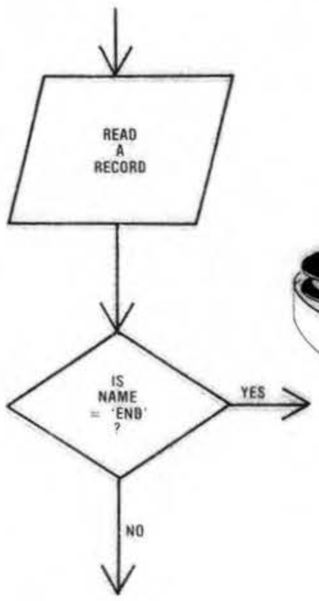
START
CLEAR THE SCREEN
PRINT 'WEEKLY PAYROLL REPORT'
PRINT 'EMPLOYEE', 'EMPLOYEE', 'HOURS', 'PAY', 'GROSS'
PRINT 'NUMBER', 'NAME', 'WORKED', 'RATE', 'PAY'
SET TOTAL GROSS PAY TO Ø
G1
READ A RECORD
IS NAME = 'END' ?
YES
PRINT 'TOTAL PAYROLL', TOTAL GROSS PAY
END
NO
ARE HOURS GREATER THAN 4Ø ?
YES
REGULAR PAY = 4Ø × RATE
OVERTIME PAY = (HOURS − 4Ø) × (RATE × 1.5)
GROSS PAY = REGULAR PAY + OVERTIME PAY
A2
NO
GROSS PAY = HOURS × RATE
A2
A2
ADD GROSS PAY TO TOTAL GROSS PAY
PRINT NUMBER, NAME, HOURS, RATE, GROSS PAY
G1

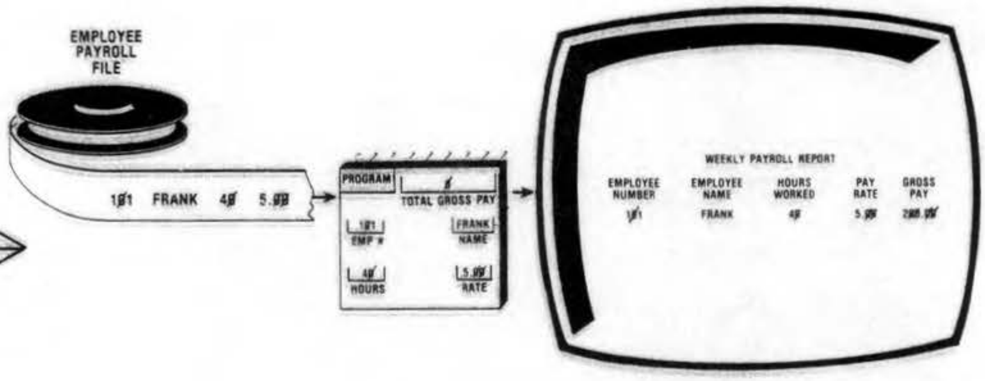
By now we have become familiar with our initialization routine: the screen is cleared, the headings are printed, and the accumulation counter is set to zero. Prior to entering the loop, our computer system looks like this:



```
                 WEEKLY PAYROLL REPORT
EMPLOYEE  EMPLOYEE   HOURS    PAY    GROSS
 NUMBER     NAME     WORKED   RATE    PAY
```

Next, a record is read. Notice that in this flowchart, a more accurate test is made for the trailer record—that is, the trailer record will contain the value 'END' where the employee's name would normally be. Thus, in our decision box, we ask the question "Is the name *equal to* 'END'?" If the answer is yes, a branch will occur to the right; otherwise, the next statement will be executed.





```
                 WEEKLY PAYROLL REPORT
EMPLOYEE  EMPLOYEE   HOURS    PAY    GROSS
 NUMBER     NAME     WORKED   RATE    PAY
   1Ø1     FRANK       4Ø     5.ØØ   2ØØ.ØØ
```

The name in this record is FRANK, not END, so the flowchart will follow the "NO" path.

At this point, in order to compute the gross pay of each employee, it is necessary to determine whether or not overtime hours were worked. The question is then asked: "ARE HOURS GREATER THAN 4Ø?" Each record's "hours" field will be compared to the constant, numeric value "4Ø" and one of two possible answers will be returned. If more than 4Ø hours were worked, the logic will branch to the right; if 4Ø or less hours were worked, no branch will occur.

According to our input file, Frank worked exactly 4Ø hours (which, of course, is not *greater than* 4Ø), so his pay will be figured out from the box "A2" shown below.

PROGRAMMER'S TIP

< means "less than"

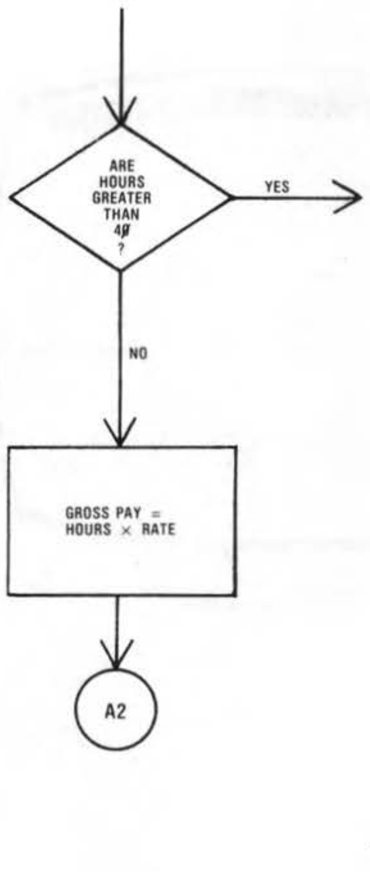
while

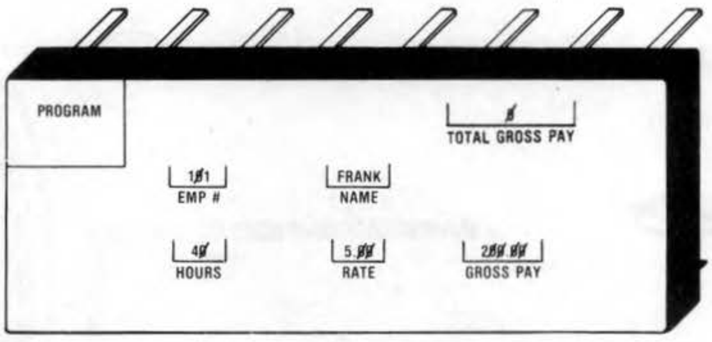
> means "greater than"

and

= means "the same as"

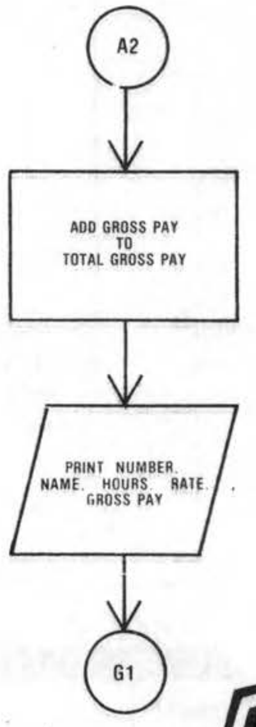
FIGURE 6—Symbols such as < , > and = make it easier and faster to say what you mean.

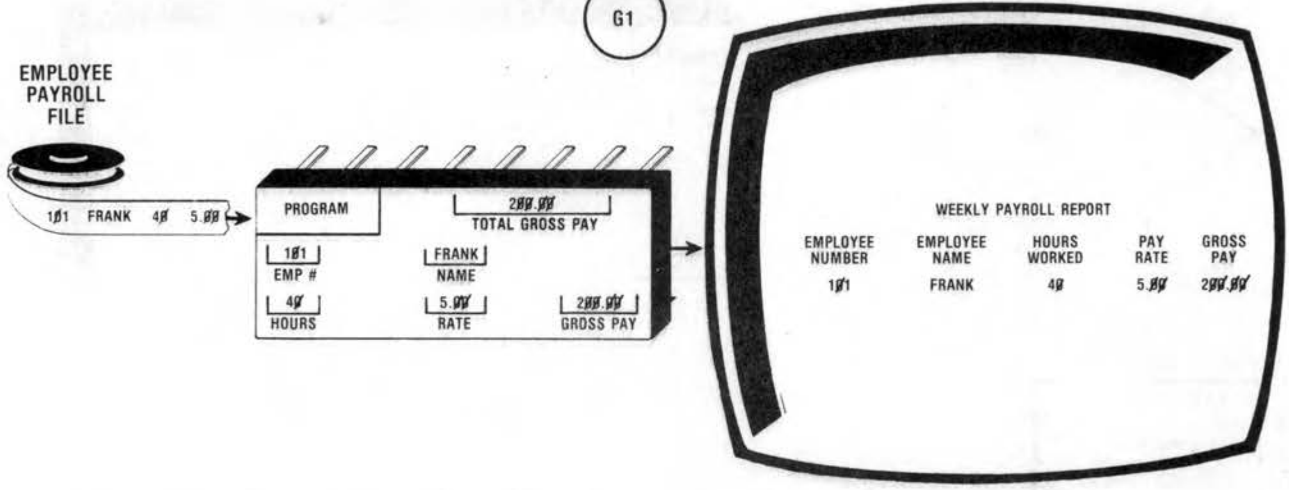




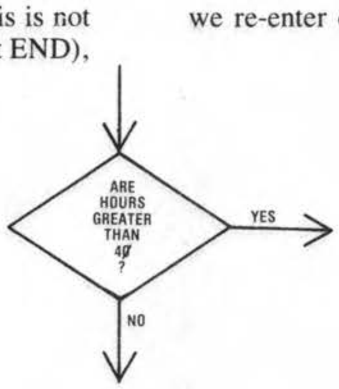
```
   4Ø
 × 5.ØØ
 2ØØ.ØØ
```

Following the flow via the connector symbol to box "A2," the just-calculated gross pay is added to the total gross pay accumulator and a detail line is printed as seen below:

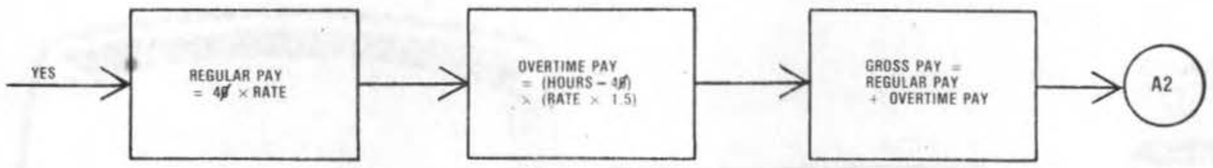

A2
ADD GROSS PAY TO TOTAL GROSS PAY
PRINT NUMBER, NAME, HOURS, RATE, GROSS PAY
G1



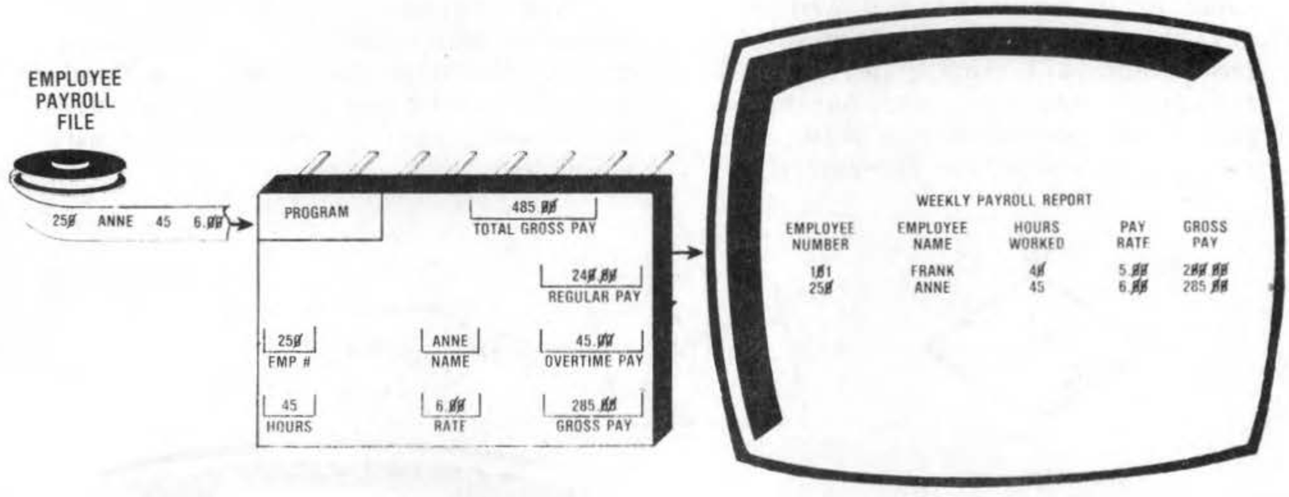
Then, the next record is read. As this is not the trailer record (the name is ANNE, not END), we re-enter our comparing logic.



When the decision box is executed this time, "hours" is found to be greater than 4Ø and the path to the right is taken.

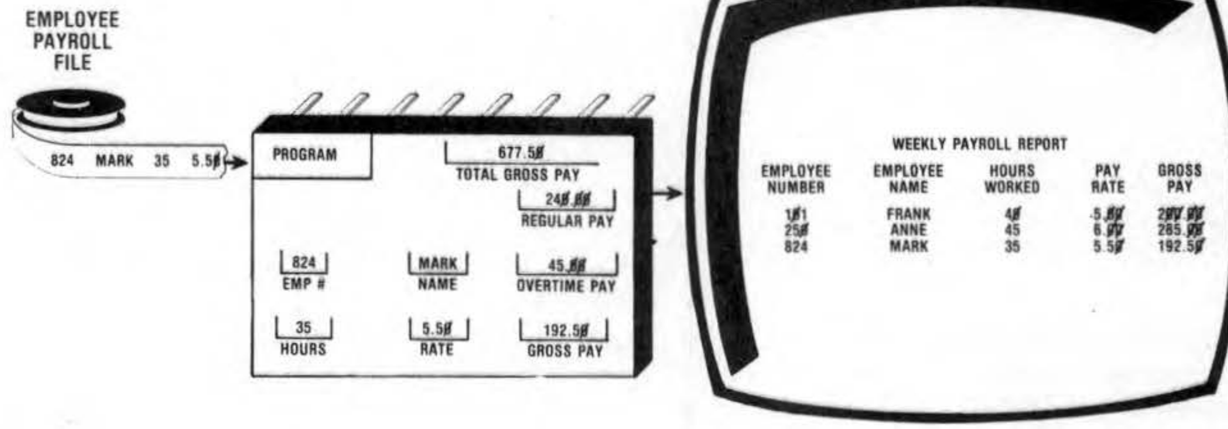

YES
REGULAR PAY = 4Ø × RATE
OVERTIME PAY = (HOURS − 4Ø) × (RATE × 1.5)
GROSS PAY = REGULAR PAY + OVERTIME PAY
A2

Notice that in these calculations, the gross pay is the sum of the regular pay (given for the first 4Ø hours worked) and the overtime pay (one-and-a-half times the regular rate of pay times the number of hours worked *over* 4Ø). In this case, the regular pay is \$24Ø.ØØ (4Ø × 6.ØØ); the overtime pay is (45.ØØ−4Ø) × (6.ØØ × 1.5) = \$45.ØØ. Once the gross pay is determined, we connect back to the main stream of the flowchart, causing the addition to the gross pay accumulator and the printing of another detail line. Our system now looks like this:



```
                    WEEKLY PAYROLL REPORT
EMPLOYEE     EMPLOYEE      HOURS      PAY     GROSS
 NUMBER        NAME        WORKED     RATE     PAY
   1Ø1        FRANK          4Ø       5.ØØ    2ØØ.ØØ
   25Ø        ANNE           45       6.ØØ    285.ØØ
```

Now, see if you can follow the steps that our third record will take, after which a third detail line will be added.

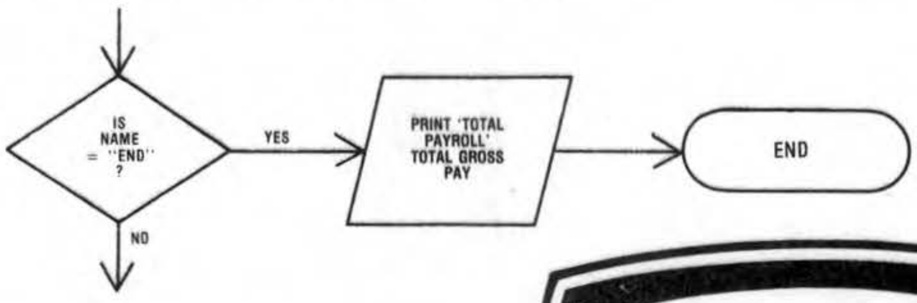


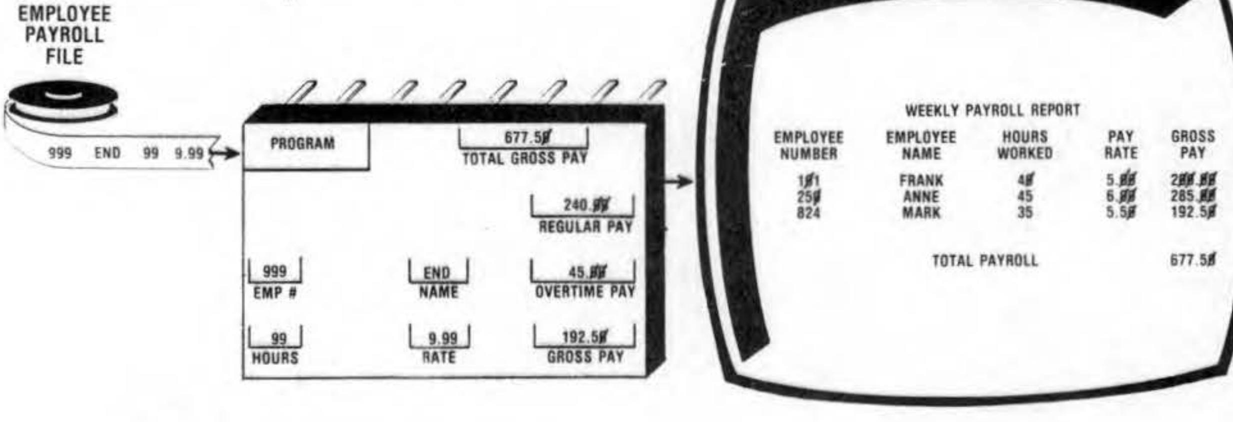
```
                      WEEKLY PAYROLL REPORT
EMPLOYEE   EMPLOYEE    HOURS     PAY    GROSS
 NUMBER      NAME      WORKED    RATE    PAY
  1Ø1       FRANK       4Ø       5.ØØ   2ØØ.ØØ
  25Ø       ANNE        45       6.ØØ   285.ØØ
  824       MARK        35       5.5Ø   192.5Ø
```

In this case, the hours were found to be less than 4Ø, the "NO" path was taken, and the gross pay became $192.5Ø (35 x 5.5Ø).

> *NOTE*: Inside of RAM, there still are values for the regular pay and overtime pay. They are left over from the previous record and will remain there until reassigned a new value. Since our third record never used these fields during its cycle, no harm was done. However, if it had, strange results would have occurred. Sometimes it is necessary to set a field back to zero before it is referenced again.

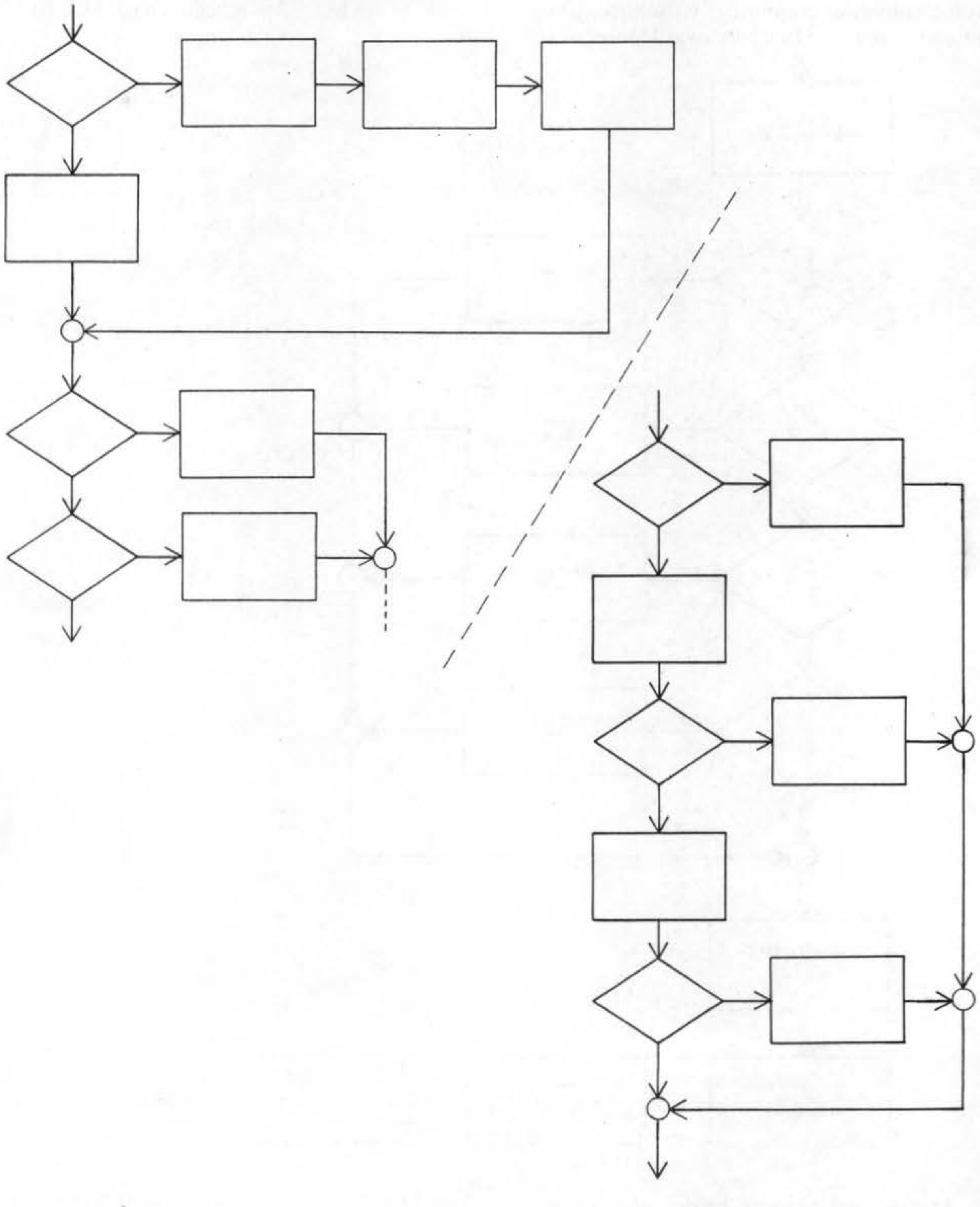
When the trailer record is read, its name field will be found equal to "END" and so the logic will flow to the right where the final total line will be printed and the program will end. The flowchart and trailer record would appear as follows:



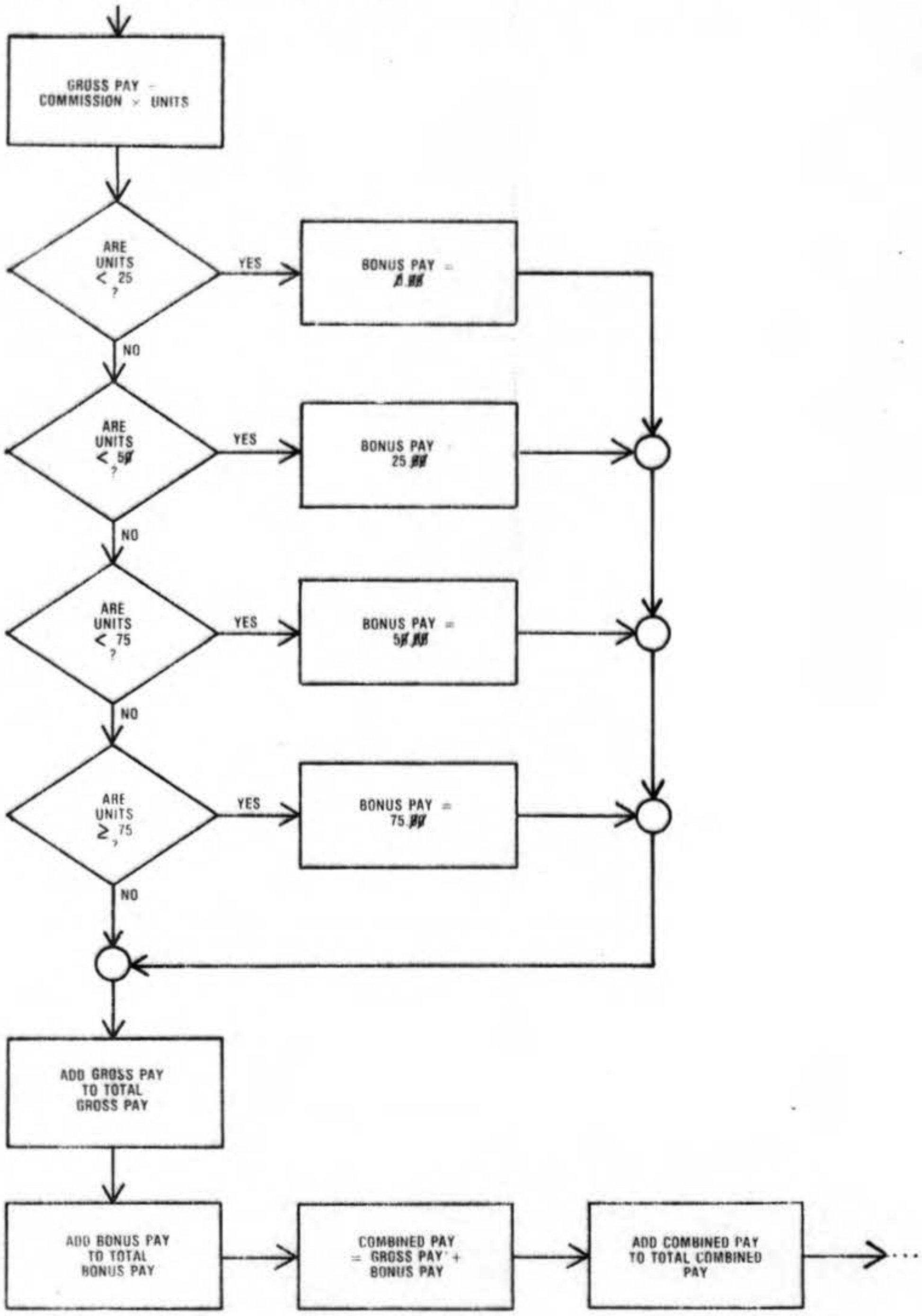


```
                      WEEKLY PAYROLL REPORT
EMPLOYEE   EMPLOYEE    HOURS     PAY    GROSS
 NUMBER      NAME      WORKED    RATE    PAY
  1Ø1       FRANK       4Ø       5.ØØ   2ØØ.ØØ
  25Ø       ANNE        45       6.ØØ   285.ØØ
  824       MARK        35       5.5Ø   192.5Ø

              TOTAL PAYROLL                677.5Ø
```

Quite often, the use of decision boxes results in choices which eventually lead to reconnecting with the major flow line of the chart. Keeping in mind that the normal flow is from top to bottom and from left to right, what happens when there are two or more options available for charting? We merely connect the lines and use small connector symbols to show such connections. Here are some examples:



The applications for using these connectors are numerous. Suppose, for example, you wanted to chart the bonuses awarded to sales people for selling more than 25, 5Ø and 75 units of product. The scheme could look like this:


GROSS PAY = COMMISSION × UNITS
ARE UNITS < 25 ?
YES
BONUS PAY = Ø.ØØ
NO
ARE UNITS < 5Ø ?
YES
BONUS PAY = 25.ØØ
NO
ARE UNITS < 75 ?
YES
BONUS PAY = 5Ø.ØØ
NO
ARE UNITS ≥ 75 ?
YES
BONUS PAY = 75.ØØ
NO
ADD GROSS PAY TO TOTAL GROSS PAY
ADD BONUS PAY TO TOTAL BONUS PAY
COMBINED PAY = GROSS PAY + BONUS PAY
ADD COMBINED PAY TO TOTAL COMBINED PAY
...

Practice using this technique of creating options and merging the flow lines back into the major chart. Now, try using this new technique of connecting flow lines in the Programmer's Check which follows. After that, we will be ready to review some other ways our three basic computer operations can be structured to produce more complex results.

PROGRAMMER'S CHECK

4

Flowcharting Payrolls

Modify the flowchart for the employee weekly payroll program to incorporate December bonuses for each employee based upon their years of employment. Also, accumulate totals for gross pay, bonuses, and combined pay.

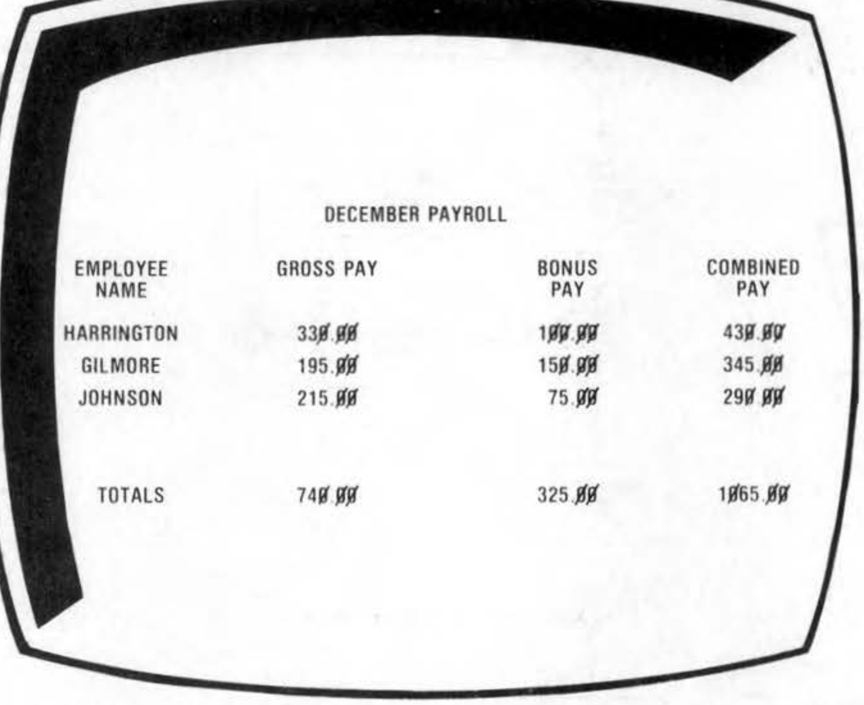
INPUT FILE

EMP. NAME	HOURS	RATE OF PAY	YEARS EMPLOYED
Harrington	5Ø	6.ØØ	2
Gilmore	3Ø	6.5Ø	4
Johnson	42	5.ØØ	1
No more	99	9.99	9

BONUS CHART

YEARS EMPLOYED	BONUS
Less than 2	75.ØØ
2-3	1ØØ.ØØ
4 or more	15Ø.ØØ

OUTPUT: CRT



```
                         DECEMBER PAYROLL

 EMPLOYEE        GROSS PAY              BONUS          COMBINED
   NAME                                  PAY              PAY

HARRINGTON        33Ø.ØØ               1ØØ.ØØ          43Ø.ØØ
  GILMORE         195.ØØ               15Ø.ØØ          345.ØØ
  JOHNSON         215.ØØ                75.ØØ          29Ø.ØØ


   TOTALS         74Ø.ØØ               325.ØØ         1Ø65.ØØ
```

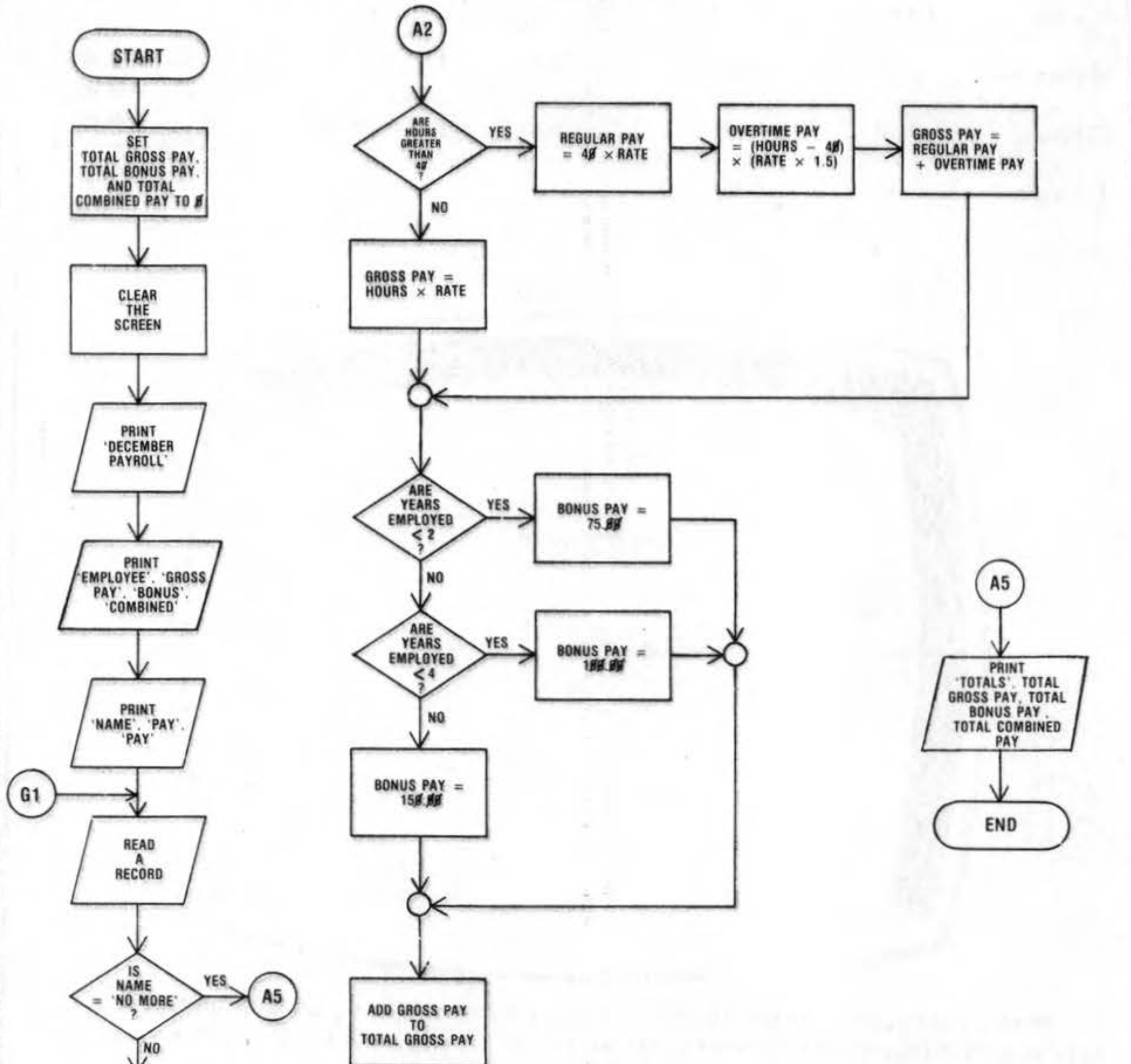
Match your solution to the one given. Yours may be different, but still valid. After you have designed your flowchart, make sure to walk it through with the input data given. As long as you get the right output, it works!

(Answers on Page 40)

PROGRAMMER'S CHECK ANSWERS

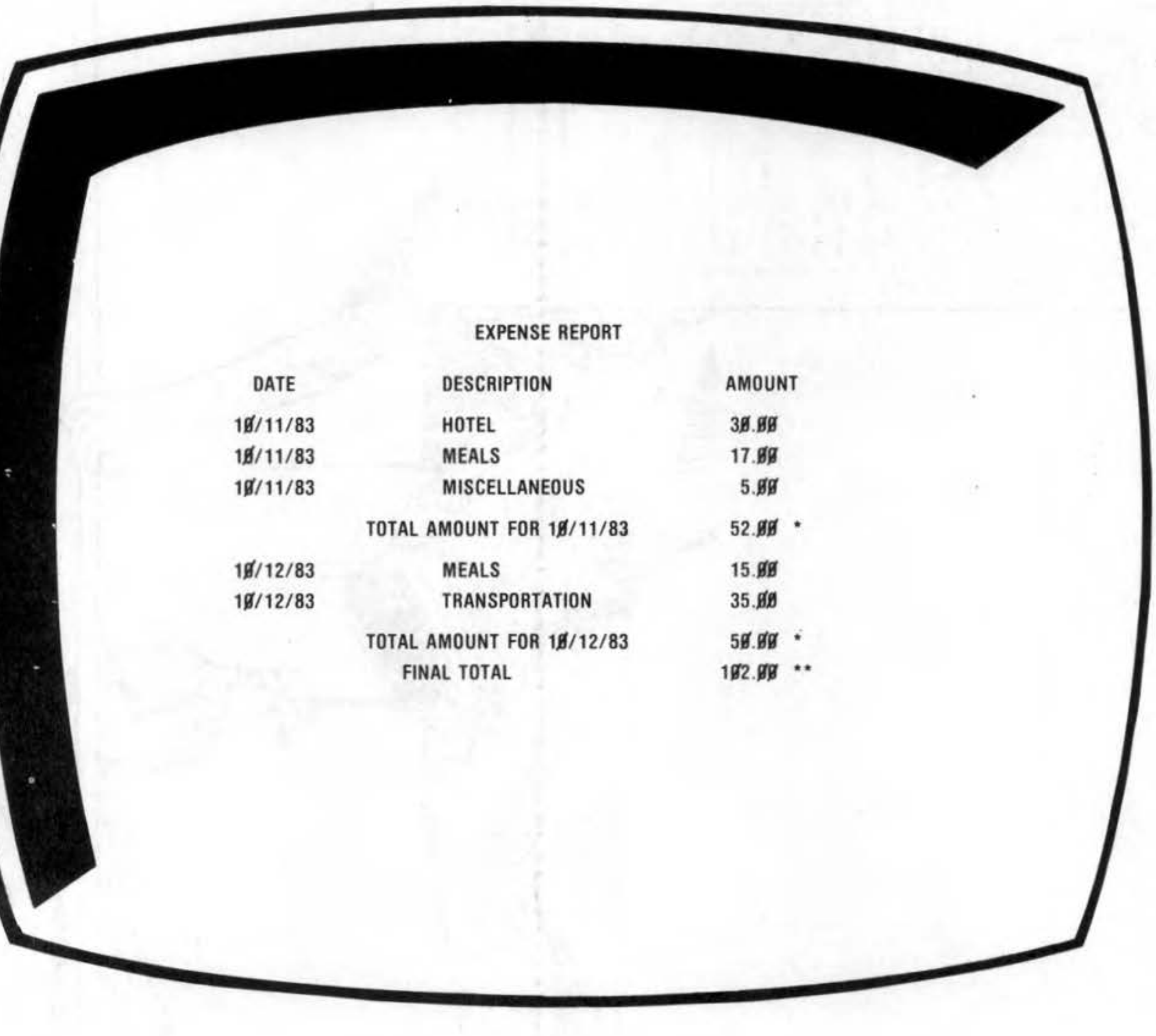
4

One Solution to Flowcharting Payrolls


START
SET TOTAL GROSS PAY, TOTAL BONUS PAY, AND TOTAL COMBINED PAY TO Ø
CLEAR THE SCREEN
PRINT 'DECEMBER PAYROLL'
PRINT 'EMPLOYEE', 'GROSS PAY', 'BONUS', 'COMBINED'
PRINT 'NAME', 'PAY', 'PAY'
G1
READ A RECORD
IS NAME = 'NO MORE' ?
YES
A5
NO
A2
A2
ARE HOURS GREATER THAN 4Ø ?
YES
REGULAR PAY = 4Ø × RATE
OVERTIME PAY = (HOURS − 4Ø) × (RATE × 1.5)
GROSS PAY = REGULAR PAY + OVERTIME PAY
NO
GROSS PAY = HOURS × RATE
ARE YEARS EMPLOYED < 2 ?
YES
BONUS PAY = 75.ØØ
NO
ARE YEARS EMPLOYED < 4 ?
YES
BONUS PAY = 1ØØ.ØØ
NO
BONUS PAY = 15Ø.ØØ
ADD GROSS PAY TO TOTAL GROSS PAY
ADD BONUS PAY TO TOTAL BONUS PAY
COMBINED PAY = GROSS PAY + BONUS PAY
ADD COMBINED PAY TO TOTAL COMBINED PAY
PRINT NAME, GROSS PAY, BONUS PAY, COMBINED PAY
G1
A5
PRINT 'TOTALS', TOTAL GROSS PAY, TOTAL BONUS PAY, TOTAL COMBINED PAY
END

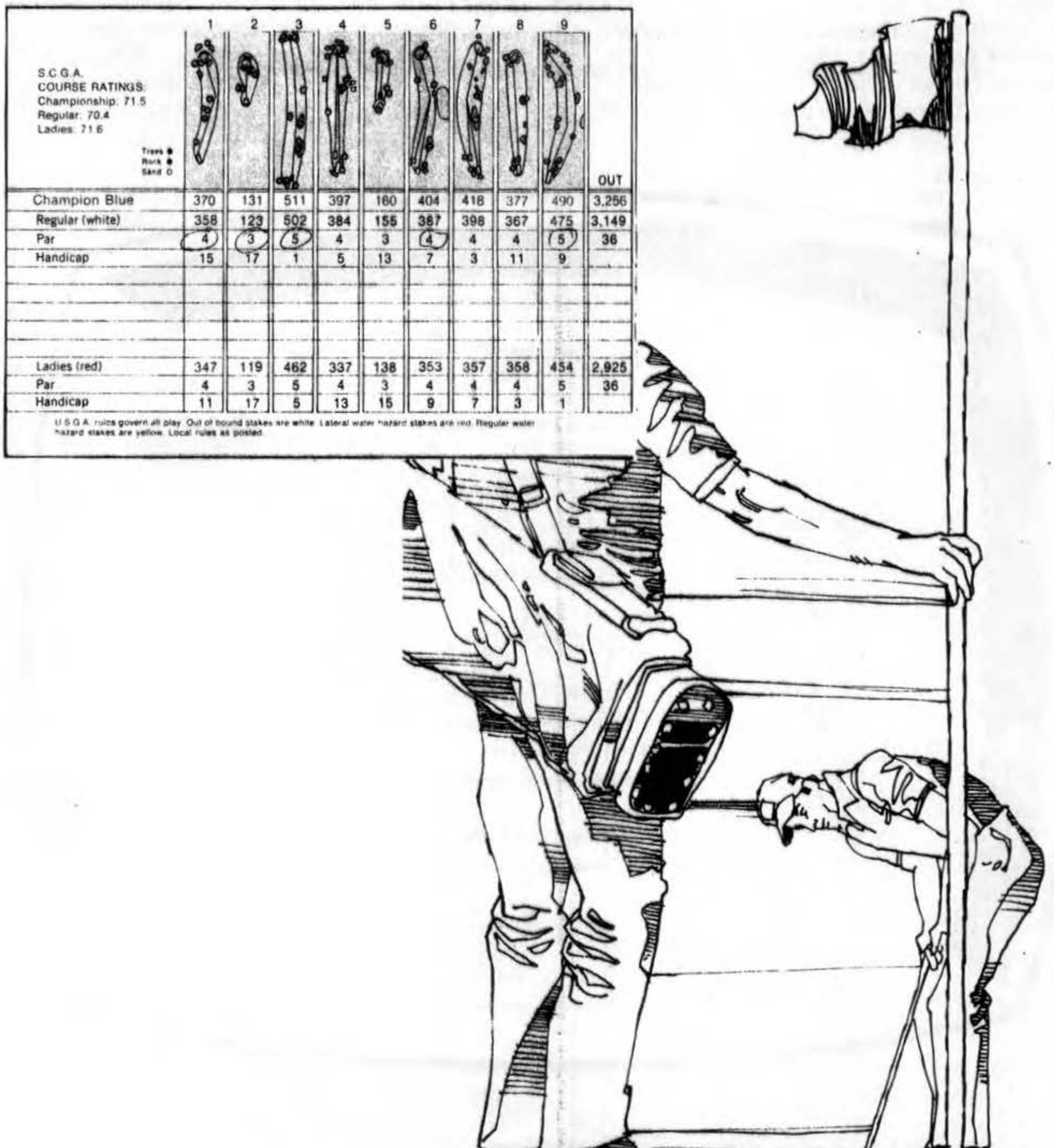
CONTROL BREAKS

We have seen how totals can be taken of records from an entire file. Sometimes, however, it is necessary to group records together within a file and to take intermediate totals for each group. Each record in the file will have a "control field" or a group identifier field. As each record is read, but before it is processed, we must compare its control field to the previous control field. In this way we are able to determine when a new group is about to be processed so that totals of the previous group can be printed. See, in the following illustration, the way a program using control break logic can display its output.



```
                         EXPENSE REPORT

   DATE              DESCRIPTION                  AMOUNT

 1Ø/11/83            HOTEL                         3Ø.ØØ
 1Ø/11/83            MEALS                         17.ØØ
 1Ø/11/83            MISCELLANEOUS                  5.ØØ

              TOTAL AMOUNT FOR 1Ø/11/83            52.ØØ  *

 1Ø/12/83            MEALS                         15.ØØ
 1Ø/12/83            TRANSPORTATION                35.ØØ

              TOTAL AMOUNT FOR 1Ø/12/83            5Ø.ØØ  *
                   FINAL TOTAL                    1Ø2.ØØ  **
```

In this example, the date serves as the control field. Each time a record of a new date is read (except for the first record), intermediate totals for the previous date are printed.



S.C.G.A.
COURSE RATINGS:
Championship: 71.5
Regular: 70.4
Ladies: 71.6

Trees ●
Rock ●
Sand ○

	1	2	3	4	5	6	7	8	9	OUT
Champion Blue	370	131	511	397	160	404	418	377	490	3,256
Regular (white)	358	123	502	384	155	387	398	367	475	3,149
Par	4	3	5	4	3	4	4	4	5	36
Handicap	15	17	1	5	13	7	3	11	9	
Ladies (red)	347	119	462	337	138	353	357	358	454	2,925
Par	4	3	5	4	3	4	4	4	5	36
Handicap	11	17	5	13	15	9	7	3	1	

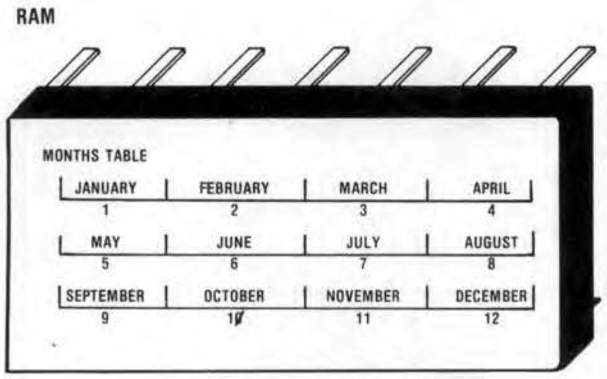
U.S.G.A. rules govern all play. Out of bound stakes are white. Lateral water hazard stakes are red. Regular water hazard stakes are yellow. Local rules as posted.

FIGURE 7—Control-break logic is used when keeping scores of professional golf matches. Totals are maintained for each nine holes in a round of 18. Then, each round is recorded before reporting the 72-hole total. You could construct a program with control breaks to record and flash results on monitors or electronic scoreboards.

As you can see, control-break logic has many applications in daily life as budgets, inventories, expense reports and other reports are tabulated. And speaking of tabulation, tables and arrays are also logic structured.

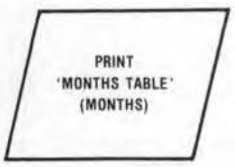
TABLES AND ARRAYS

In our previous programs, each field contained only that value just assigned to it. Many times, in data processing applications, it is desirable to maintain more than one value for a given field in RAM at the same time. A table (also known as an array) is a series of consecutive storage locations, each containing "like" data. Only one field name is assigned to the table although several values within the table may be referenced through the use of a subscript. A subscript is a whole number which indicates the position of a particular value inside of a table. For example, we can "load" (copy) a table consisting of 12 values of the names of the months into main storage.



If we needed to access the fifth month of the year, we could say MONTHS TABLE (5). Within the parentheses is the subscript which, in this case, is 5, which means the fifth position of the months table or "MAY."

Very often, tables are searched before data is extracted from them. In these cases, the subscript itself will be a variable. If, for example, a two-digit month field were read from the input record, it could serve as the subscript so that we could print the name of the month from the table as in:

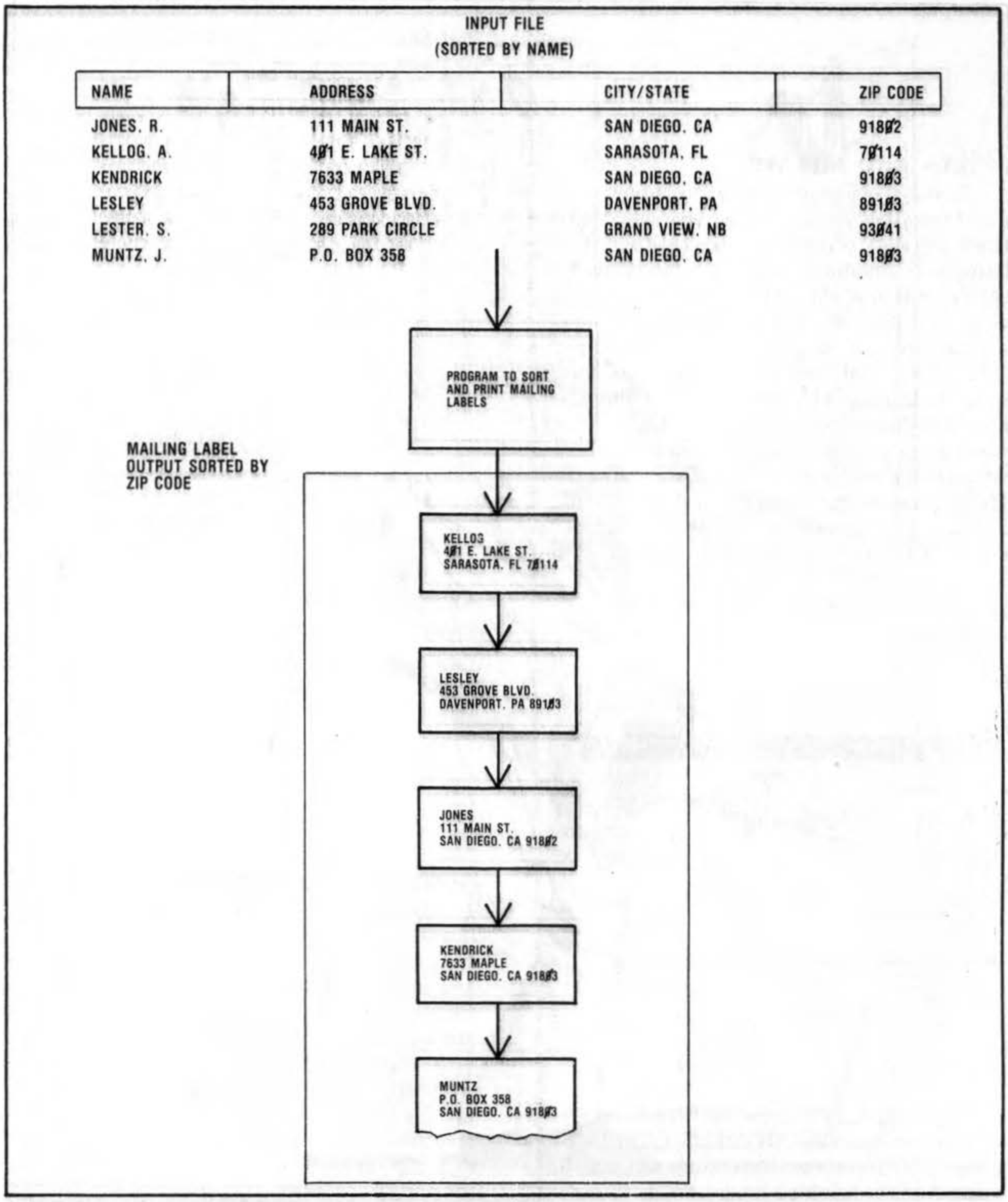


Here, if months equal 1Ø, the value "OCTOBER" would be printed.

We will later utilize tables in a way in which much information can be extracted from them through the use of variables and codes on the input.

SORTING AND MERGING

In many applications, records from an input file must be resequenced into some alphabetic or numeric order. This process is known as "sorting." For example, say our input file contains records with name, address, city, state and zip code fields. These records are in alphabetic sequence by name. If our program were to generate mailing labels from each record on the file, it might be required that we sort the records into numeric sequence according to their zip codes to facilitate their handling by the post office. Such a program might look like the one on the next page.

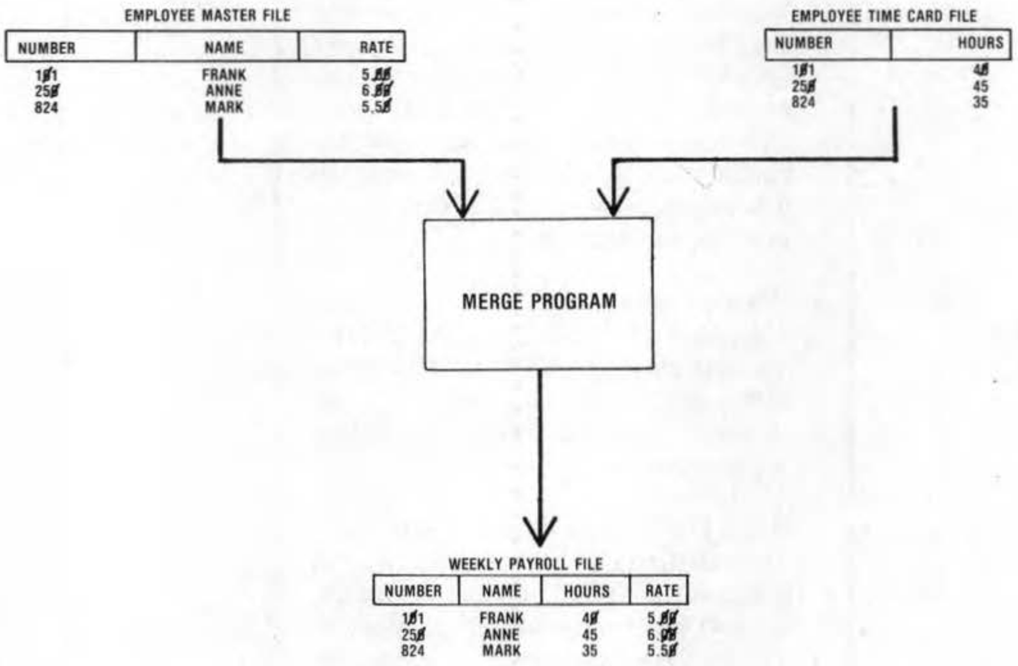

INPUT FILE
(SORTED BY NAME)

NAME	ADDRESS	CITY/STATE	ZIP CODE
JONES, R.	111 MAIN ST.	SAN DIEGO, CA	918Ø2
KELLOG, A.	4Ø1 E. LAKE ST.	SARASOTA, FL	7Ø114
KENDRICK	7633 MAPLE	SAN DIEGO, CA	918Ø3
LESLEY	453 GROVE BLVD.	DAVENPORT, PA	891Ø3
LESTER, S.	289 PARK CIRCLE	GRAND VIEW, NB	93Ø41
MUNTZ, J.	P.O. BOX 358	SAN DIEGO, CA	918Ø3

PROGRAM TO SORT AND PRINT MAILING LABELS
MAILING LABEL OUTPUT SORTED BY ZIP CODE
KELLOG 4Ø1 E. LAKE ST. SARASOTA, FL 7Ø114
LESLEY 453 GROVE BLVD. DAVENPORT, PA 891Ø3
JONES 111 MAIN ST. SAN DIEGO, CA 918Ø2
KENDRICK 7633 MAPLE SAN DIEGO, CA 918Ø3
MUNTZ P.O. BOX 358 SAN DIEGO, CA 918Ø3

Merging refers to the combining of records from more than one input file into one output file. Often, data is collected onto files and then added to another file periodically. Such a process can be demonstrated in an application involving the maintaining of an employee master payroll file.

The master file might contain data about each employee, including their hourly rate of pay. Each week records are created onto an employee time card file. This file contains the number of hours worked that week by each employee. In order to calculate the gross pay data for each employee, both files must be in main storage at the same time. One way to do this would be to merge the two files to form a third file which would look much like the one we used on our comparing application earlier in this Unit.



EMPLOYEE MASTER FILE

NUMBER	NAME	RATE
1Ø1	FRANK	5.ØØ
25Ø	ANNE	6.ØØ
824	MARK	5.5Ø

EMPLOYEE TIME CARD FILE

NUMBER	HOURS
1Ø1	4Ø
25Ø	45
824	35

WEEKLY PAYROLL FILE

NUMBER	NAME	HOURS	RATE
1Ø1	FRANK	4Ø	5.ØØ
25Ø	ANNE	45	6.ØØ
824	MARK	35	5.5Ø

When utilizing this type of system, notice how the master file can be maintained independent of the time card file, which will be changed with data collected at the end of each pay period.

Well, now you have seen the game plan—the program design. Not only that, you have also learned how to create a flowchart to show how data and processing can be handled in logical fashion by the computer. You also learned the steps of the program development cycle:

1. Analysis
2. Design
3. Coding
4. Testing and Debugging
5. Documentation

You now have a glimpse at the thought processes behind computer programming.

When you have reviewed the material and feel that you have a good understanding of what has been presented, complete the Exam which follows. Good luck!

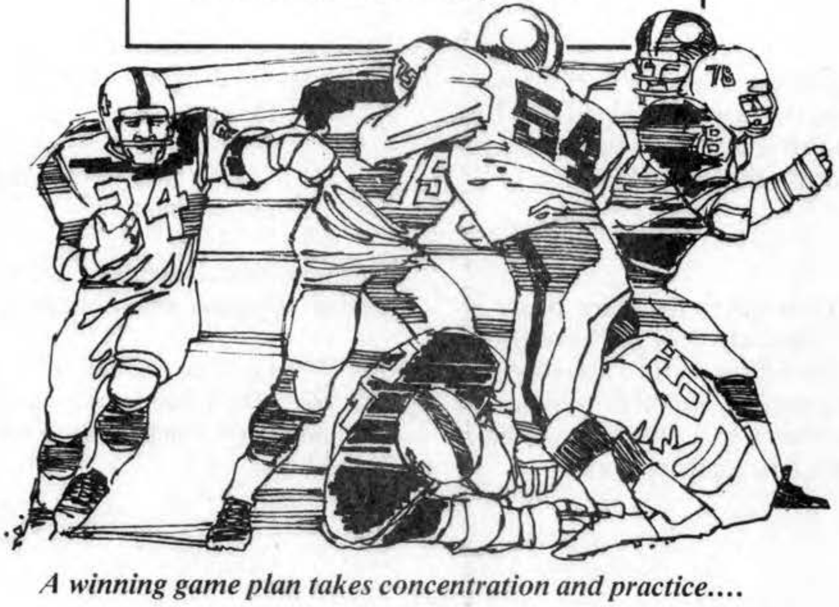
DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **How to read and design a flowchart of a program?**
  You have learned how to create a flowchart in which line items are graphically represented. You have seen how various symbols can be used to represent program and computer functions. And, most important, you have learned that program flow depends upon the use of certain conventions and procedures.

- **What a decision box is?**
  A decision box is a diamond-shaped symbol providing for alternative program paths. It provides the computer with "Yes/No" choices based upon values which are equal or unequal.

- **WHAT is the initialization Routine?**
  The initialization routine is a set of steps used to prepare the computer for receiving and processing data. Such steps as clearing the screen, printing headings, setting counters, and setting input locations become automatic with practice.



A winning game plan takes concentration and practice....

SCHOOL OF COMPUTER TRAINING

EXAM 4

PROGRAM DESIGN
THE GAME PLAN

4704-2

When you feel confident that you have mastered the material in Study Unit 4, complete Exam 4. When you have completed the entire Exam, transfer your answers to the Answer Sheet which follows.

Questions 1-20: *Circle the letter beside the one best answer to each question (5 points each).*

1. The first entry on a flowchart is:
   (a) CLEAR THE SCREEN
   (b) PRINT
   (c) READ A RECORD
   (d) START

2. If you were charting a program in which there is "branching," you would need a
   (a) display symbol.
   (b) terminal interrupt symbol.
   (c) decision box symbol.
   (d) merge symbol.

3. From largest to smallest, you have
   (a) file, record, field.
   (b) record, file, field.
   (c) field, record, file.
   (d) field, file, record.

4. READ A RECORD would require a
   (a) processing box.
   (b) connector symbol.
   (c) decision box.
   (d) input/output box.

5. The symbol used whenever data must be read into computer memory from a keyboard or auxiliary storage would be
   (a) input/output symbol.
   (b) decision box.
   (c) connector symbol.
   (d) processing box.

6. The two symbols which always have one line leading into them and one line coming out are
   (a) decision boxes and connectors.
   (b) connectors and input/output boxes.
   (c) processing boxes and input/output boxes.
   (d) input/output boxes and decision boxes.

7. CLEAR SCREEN is one of several commands given during the
   (a) "branching" operation.
   (b) process of setting counters to zero.
   (c) initialization routine.
   (d) input/output stage.

8. One of the conventions used in flowcharting is that the flow of a program should be
   (a) from right to left and bottom to top.
   (b) from bottom to top and left to right.
   (c) from left to right and bottom to top.
   (d) from top to bottom and left to right.

9. When it is necessary to produce sub or intermediate totals for groups of records, a program is produced using
   (a) control-break logic.
   (b) tables and arrays.
   (c) sorting and merging.
   (d) subscripting the trailers.

10. $A > B$ means
    (a) B is greater than A.
    (b) A is greater than B.
    (c) A is equal to B.
    (d) A represents B.

QUESTIONS 11-20 REFER TO THE FOLLOWING PROGRAMMING PROBLEM.

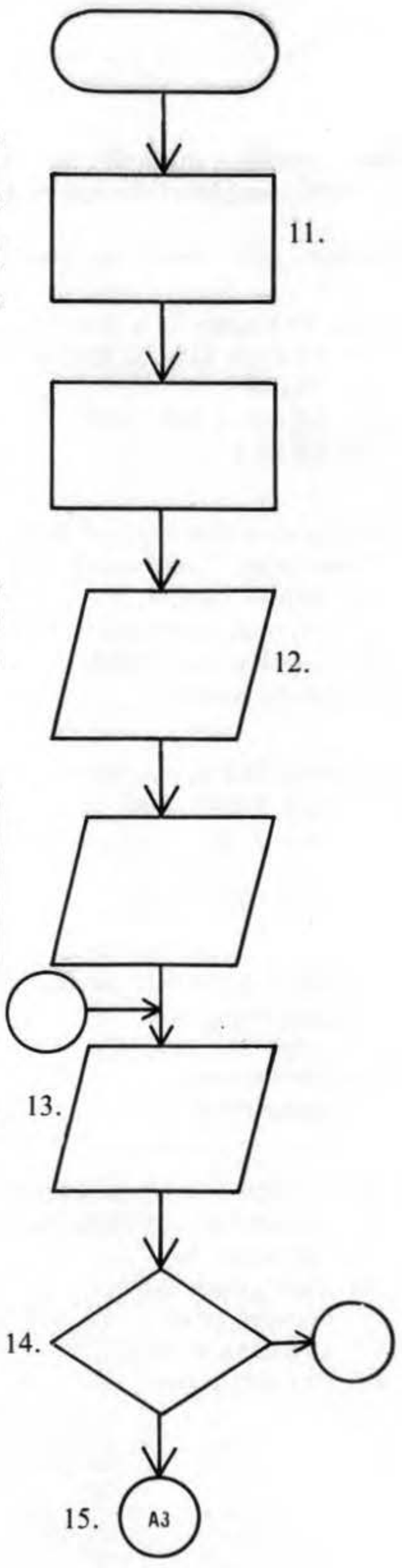
Designing a Program for an Accounts Receivable Report Application

Allen Smith has a television repair business. He has asked you to develop a program for his microcomputer so that he can track his accounts receivable. He needs to know the amount due from each customer and would also like to have a record of total amount due. Since all repairs are basically the same, he is not interested in maintaining unit or quantity figures. Below is a portion of the accounts receivable file he has provided to you for the purposes of design and flowcharting:

6-1-83	J. Echols	461 State St.	$ 57.9∅
6-1-83	E. Garrett	28∅1 Church Rd.	198.22
6-1-83	B. Jeffries	RR #2	24.75
6-1-83	W. Mandrake	5773 Park Place	19.9∅

Using the flowchart symbols provided, fill in the appropriate commands and information in the spaces. Once you have completed the chart and shown the output in the form of an "Accounts Receivable Report" on the CRT, answer the 10 questions which follow.

```
                              ACCOUNTS RECEIVABLE REPORT
DATE                  NAME                    ADDRESS                      AMOUNT DUE




                                                   TOTAL AMOUNT DUE
```

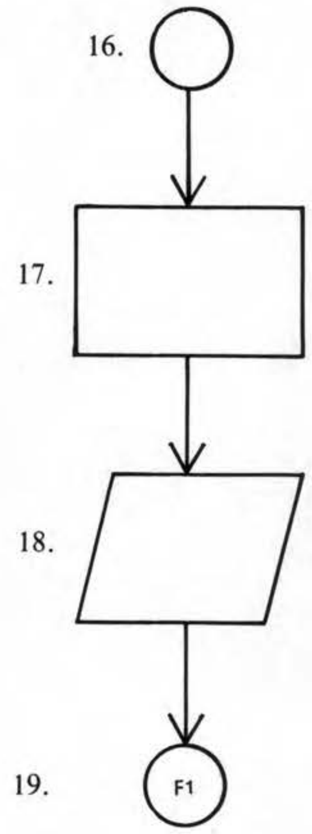


11. This box (No. 11 at right) would contain which of the following commands?

    (a) SET AMOUNT DUE TO ∅
    (b) SET TOTAL AMOUNT DUE TO ∅
    (c) PRINT 'ACCOUNTS RECEIVABLE REPORT'
    (d) PRINT 'A RECORD'

12. This box would contain:

    (a) PRINT 'ACCOUNTS RECEIVABLE REPORT'
    (b) PRINT 'A RECORD'
    (c) SET AMOUNT DUE TO ∅
    (d) AMOUNT DUE = AMOUNT DUE



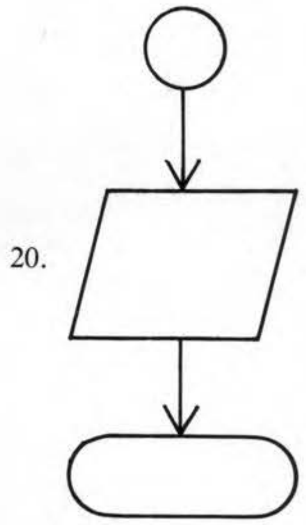
13. This box would contain:
    (a) PRINT 'ACCOUNTS RECEIVABLE REPORT'
    (b) PRINT 'TOTAL AMOUNT DUE', 'TOTAL AMOUNT DUE'
    (c) READ 'A RECORD
    (d) PRINT 'A RECORD'

14. This box
    (a) asks a question.
    (b) checks for the trailer record.
    (c) creates a loop.
    (d) all of the above.

15. A3 is
    (a) a connector to process more records.
    (b) a signal to end the program.
    (c) a connector to CLEAR SCREEN.
    (d) a symbol for END.

16. This is a symbol for
    (a) input/output.
    (b) data processing.
    (c) decisions.
    (d) none of the above.

17. The contents of this box would include:
    (a) PRINT 'DATE', 'NAME', 'ADDRESS', 'AMOUNT DUE'
    (b) AMOUNT DUE = AMOUNT DUE
    (c) TOTAL AMOUNT DUE = TOTAL AMOUNT DUE + AMOUNT DUE
    (d) READ A RECORD



18. This box is a symbol for
    (a) input/output.
    (b) data processing.
    (c) decisions.
    (d) CRT monitor.

19. The contents of the box preceding F1 would include the following command:
    (a) TOTAL AMOUNT DUE = TOTAL AMOUNT DUE + AMOUNT DUE
    (b) PRINT 'ACCOUNTS RECEIVABLE RECORD'
    (c) PRINT DATE, NAME, ADDRESS, AMOUNT DUE
    (d) READ A RECORD

20. The command in this box would include:
    (a) PRINT 'TOTAL AMOUNT DUE', TOTAL AMOUNT DUE
    (b) PRINT 'ACCOUNTS RECEIVABLE REPORT'
    (c) PRINT DATE, NAME, ADDRESS, AMOUNT DUE
    (d) AMOUNT DUE = AMOUNT DUE

SCHOOL OF COMPUTER TRAINING

ANSWER SHEET

PROGRAM DESIGN—THE GAME PLAN

Examination Number 4704-2

PLEASE PRINT

NAME ______________________ STUDENT NUMBER ______________

ADDRESS ____

CITY ______________________ STATE ________ ZIP ______________

☐ **Check if this is a new address and you have not previously notified us.**

— FOLD —

CUT ALONG THIS LINE

INDICATE YOUR ANSWER TO EACH QUESTION BY MARKING AN **X** IN THE APPROPRIATE SQUARE. EXAMPLE: [✓ A] B C D

1.	A	B	C	D	**11.**	A	B	C	D
2.	A	B	C	D	**12.**	A	B	C	D
3.	A	B	C	D	**13.**	A	B	C	D
4.	A	B	C	D	**14.**	A	B	C	D
5.	A	B	C	D	**15.**	A	B	C	D
6.	A	B	C	D	**16.**	A	B	C	D
7.	A	B	C	D	**17.**	A	B	C	D
8.	A	B	C	D	**18.**	A	B	C	D
9.	A	B	C	D	**19.**	A	B	C	D
10.	A	B	C	D	**20.**	A	B	C	D

— FOLD —